



UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Testes automáticos de acessibilidade em aplicações móveis

José Rui Monteiro Chantre

Tese para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Simão Melo de Sousa

Covilhã, Outubro de 2015

Agradecimentos

Em primeiro lugar, agradecer a minha família que sempre estiveram do meu lado e de forma incondicional. Sem a encorajamento e incentivo não teria sido possível chegar a este ponto.

Quero agradecer também ao do Prof. Doutor Simão Melo de Sousa, que pelas suas linhas de orientação foi possível desbravar caminho, e em muitas vezes sair de "encruzilhadas" inerentes a atividade de procura do conhecimento.

A empresa Altran que me proporcionou as condições necessárias para desenvolver uma dissertação de mestrado no meio de um ambiente empresarial. A toda equipa de testes da Axa Banque, no Fundão, em especial Ana Ramos que através dos seus conhecimentos na área, contribui imenso com ideias e sugestões. Ao Hugo Firmino, que com a sua visão pragmática de gestão de projetos e no estabelecimento de metas.

Gostaria de agradecer também aos meus amigos Manoel Campos e Rayssa Campos pelo apoio e ajuda. A Cristina que, incondicionalmente, esteve do meu lado.

E por último, agradecer aos meus colegas do RELiable And SEcure Computation Group (RELEASE) pelas constantes trocas de ideias, o que proporcionou, em muitos casos o surgimento de novas soluções.

"Se soubessemos o que estamos a fazer não se chamaria investigação". - Albert Einstein

Resumo

Resumo do trabalho

A presente dissertação resulta da necessidade do desenvolvimento de uma ferramenta para automatização dos testes de acessibilidade das aplicações móveis, com base nas normas estabelecidas pelos organismos internacionais.

Esta dissertação está inserida no contexto empresarial, pelo que, visam aumentar os serviços de testes disponibilizadas pela empresa Altran ao cliente Axa Banque.

Nesta dissertação é apresentado o estudo dos métodos e técnicas dos testes de software tradicionais e a sua aplicação no contexto das aplicações móveis e um estudo sobre as normas de acessibilidade definidas pela World Wide Web Consortium e pela legislação Section 508 dos Estados Unidos da América.

A ferramenta desenvolvida segue os princípios de engenharia de software definidas pela metodologia ICONIX e permite a execução de testes de acessibilidade em aplicações móveis, e posteriormente, a integração dos resultados em plataformas de gestão de testes, como o Testlink. Como caso de estudo foi utilizada a aplicação do banco francês Axa Banque que está inserida na categoria de Home Banking.

Palavras-chave

Testes de software, Acessibilidade, Usabilidade, Aplicações móveis

Abstract

This work results from the need to develop a tool for automation of accessibility testing for mobile applications based on standards that are set by international organizations.

This work is applied to the business context and aims to increase the testing services provided by the company Altran to its customer Axa Banque.

This thesis presents a study of methods and techniques for traditional software testing and its application in the context of mobile applications and a study on the accessibility standards defined by the World Wide Web Consortium and Section 508 of the legislation of the United States of America.

The developed tool follows the principles of software engineering from the ICONIX methodology and allows the execution of accessibility testing in mobile applications, and subsequently the integration of the results in test management platforms such as Testlink. The application of the French bank Banque Axa that is included in the category of home banking, was used as a case study.

Keywords

Software testing, Accessibility, Usability, Mobile application

Índice

1	Introdução	1
1.1	Enquadramento	1
1.2	Objetivo da dissertação	2
1.3	Motivação	2
1.4	Abordagem	2
1.5	Estrutura do documento	3
2	Estado de Arte dos Testes de Software	5
2.1	Introdução	5
2.2	Testes de Software	5
2.2.1	A importância dos testes	6
2.2.2	Conceitos e Terminologias	6
2.2.3	O processo de testes	7
2.2.4	Quando e onde começar os testes	8
2.2.5	Métodos e técnicas de Teste	9
2.2.6	Níveis de Teste	10
2.2.7	Testes manuais e testes automatizados	12
2.3	Testes de Aplicações móveis	13
2.3.1	Tipos de aplicações móveis	13
2.3.2	Diferenças entre testes móveis e testes clássicos	13
2.3.3	Desafios nos testes de aplicações móveis	14
2.3.4	Estratégias de testes de aplicações móveis	15
2.4	Ferramenta de gestão de testes Testlink	15
2.5	Trabalhos Relacionados	16
2.6	Conclusão	18
3	Teste de Acessibilidade Digital	19
3.1	Introdução	19
3.2	Acessibilidade num contexto europeu	19
3.3	As Normas de Acessibilidade digital	20
3.4	<i>Web Content Accessibility Guidelines</i> (WCAG)	21
3.4.1	WCAG 1.0	21
3.4.2	WCAG 2.0	22
3.5	<i>Mobile Web Best Practices</i> (MWBP)	24
3.6	Section 508	25
3.7	Relação entre Section 508 e as diretrizes da W3C	26
3.8	O porquê da acessibilidade nas aplicações	27
3.9	A abordagem para Automatizáveis	28
3.10	Tipos de Testes de acessibilidade	29
3.11	Conclusão	29

4	Análise, arquitetura e metodologia de desenvolvimento	31
4.1	Introdução	31
4.2	Metodologia de desenvolvimento ICONIX	31
4.3	Análise de requisitos	33
4.3.1	Classificação dos Requisitos	34
4.3.2	Diagrama de casos de uso	36
4.3.3	Modelo de domínio	36
4.4	Análise e desenho preliminar	37
4.4.1	Análise de Robustez	40
4.5	Desenho detalhado	42
4.5.1	Especificar o comportamento	42
4.5.2	Arquitetura geral da ATAT	44
4.5.3	Diagrama de Classes	46
4.6	Conclusão	48
5	Implementação da ferramenta ATAT	49
5.1	Introdução	49
5.2	Implementação	49
5.2.1	Implementação do cliente	49
5.2.2	Implementação do servidor	49
5.2.3	Principais ferramentas utilizadas	52
5.3	Criação e execução de testes automáticos	52
5.3.1	Preparação dos testes	52
5.3.2	Criação de um plano de testes de acessibilidade	54
5.3.3	Execução dos testes	55
5.4	Resultados dos testes acessibilidade do caso de estudo	56
5.4.1	Testes manuais para provar o funcionamento	58
5.4.2	Resultados detalhados dos testes de acessibilidade	59
5.5	Conclusão	61
6	Conclusão	63
6.1	Trabalhos futuros	63
	Bibliografia	65
A	Anexos	73
A.1	Datasheets dos componentes utilizados	73

Lista de Figuras

2.1	Modelo de dados de entrada e saída de um sistema em testes. (adaptado de [Som10])	5
2.2	Fases de desenvolvimento de um sistema de software. Fonte: (Burnstein2002). .	8
2.3	Relação entre custo e fase de teste. Adaptado de [Qua10].	9
2.4	Granularidade dos testes de software. Adaptado de [CYZC].	11
2.5	Diagrama de casos de uso da Testlink (adaptado de [dSV10])	16
2.6	Modelo unificado para avaliar acessibilidade e usabilidade Fonte: [BBC ⁺ 10] . . .	18
3.1	As normas de acessibilidade (Fonte: W3C)	20
3.2	Cronologia das versões de WCAG	21
3.3	A estrutura da WCAG 2.[W3C08]	23
3.4	Descrição dos níveis de acessibilidade da WCAG 2.0	23
3.5	Relação entre WCAG 2.0 e MWBP 1.0	24
4.1	Diagramas da visão estática e dinâmica do ICONIX (adaptado de [dSV10])	32
4.2	Visão geral da metodologia ICONIX (Fonte: [dSV10])	33
4.3	Processo de Engenharia de requisitos (adaptado de [Jal08])	34
4.4	Visão geral das funcionalidades do sistema	36
4.5	Modelo de domínio	37
4.6	Importância dos diagramas de robustez	40
4.7	Os objetos da diagrama de robustez	40
4.8	Diagrama de robustez para criação de projetos	41
4.9	Diagrama de robustez para criação	42
4.10	Diagrama de sequência do caso de uso “Executar teste”.	43
4.11	Diagrama de sequência da criação de um plano de teste.	44
4.12	Figura XX: Arquitetura geral do sistema(a).	45
4.13	Arquitetura geral do sistema (b).	46
4.14	Diagrama de classes parcial.	47
5.1	Arquitetura do servidor de testes.	50
5.2	Comunicação bidirecional	51
5.3	Página inicial do Testlink	53
5.4	Criação de um projeto de testes completo	53
5.5	Criação de um projeto de testes completo	54
5.6	Criação de teste suite no <i>Testlink</i>	54
5.7	Criação de teste suite no Testlink	55
5.8	Interface Test Runner	55
5.9	Aplicação da Axa Banque	56
5.10	Login à aplicação da Axa Banque	56
5.11	Logins	57
5.12	Exemplo de execução testes manualmente	59

Lista de Tabelas

3.1	Distribuição das guidelines da WCAG 2.0 (Fonte: W3C)	23
3.2	Relação entre MWBP e WCAG (adaptado de [Cen14])	25
3.3	Relação entre Section 508 e a WCAG 1.0	27
3.4	O porque da acessibilidade	27
3.5	Relação entre WCAG 2.0 e Section508	28
4.1	Requisitos funcionais	35
4.2	Descrição do caso de uso “Criar Plano de testes” de acessibilidade	38
4.3	Descrição do caso de uso “Executar testes”	39
4.4	Criação de Projeto de testes	39
5.1	Resultados gerais dos testes de acessibilidade do caso de estudo	58
5.2	Resultados da execução manual	58
5.3	Resultados detalhados dos testes	59
5.4	Tempo de execução dos testes	61

Lista de Acrónimos

W3C	World Wide Web Consortium
RUP	Rational Unified Process
ER	engenharia de requisitos
UML	Unified Model Language
PHP	Hypertext Preprocessor
XP	eXtreme Programmimg
HTTP	Hypertext Transfer Protocol
GUI	Graphical User Interface
XML	EXtensible Markup Language
POJO	Plain Old Java Object
API	Application Programming Interface
DAO	Data Access Object
MVC	Model View Controller
RPC	Remote Precedure Call
JSON	JavaScript Object Notation
WS	Web Service
SOAP	Simple Object Access Protocol
ESA	European Space Agency
AUT	Aplication Under Test
S.M.A.R.T.	Specific Measurable Achievable Relevant Time-bound
TDD	Test Driven Development
ATAT	Acessibility Tests Automation Tool
WAI	Web Accessibility Iniciative
WCAG	Web Content Accessibility Guidelines
MWBP	Mobile Web Best Practices

Capítulo 1

Introdução

1.1 Enquadramento

Atualmente as tecnologias de informação e a Internet adquiriram uma importância vital na sociedade moderna, tornando-se indispensáveis à vida quotidiana, sendo que 45% da população mundial dispõe de ligação à Internet, o que significa um crescimento de 806% de 2000 a 2015 [Wor15]. Este crescimento deve-se em parte ao aumento da procura de dispositivos móveis, inicialmente dos *smartphones* e posteriormente dos *tablets*. A acompanhar esse crescimento, está o aumento explosivo da oferta de aplicações móveis disponíveis nas lojas online.

Inicialmente as aplicações móveis eram desenvolvidas para fins informativos e de produtividade e incluíam o acesso ao e-mail, contactos, calendário e informações meteorológicas. Porém, a sua ubiquidade e popularidade rapidamente permitiram a sua expansão para outros domínios e sectores de atividade, nomeadamente, saúde, finanças, seguros, entretenimento, media, educação, e o sector bancário. Estes sectores, considerados cruciais para qualquer economia, representam um mercado amplo a ser explorado [KK13].

Segundo o relatório da Clearwater Technology, as mais valias da indústria das aplicações móveis em 2015 será de 330 biliões de dólares [CF14]. Estes números são corroborados pelo estudo da *Internacional Data Corporation* (IDC) que prevê que até ao final de 2015, já terão sido transferidos da Internet cerca de 182,7 mil milhões de aplicações móveis, o que significa um aumento de 1.600% em relação aos 10,7 biliões de aplicações transferidas em 2010 [Ute15]. No entanto, este aumento exponencial mencionado anteriormente, conduziu à diminuição da qualidade das aplicações, em cerca de 44% para o mesmo período, o que abre uma janela de oportunidade ao surgimento de ferramentas de testes automatizados para aplicações móveis [IKK14].

Criar conteúdos Web e aplicações móveis que funcionem de acordo com o esperado no maior número de dispositivos móveis possível, constitui um desafio para a maioria dos criadores de conteúdos e aplicações. Neste contexto e englobado na estratégia da Altran para a criação de um centro de testes de excelência na sua filial em Portugal, surge o tema desta dissertação: *Mobile Testing*. De modo mais específico, esta dissertação está inserida nos testes das aplicações móveis das aplicações do banco Axa Banque, a qual, se definiu como objetivo o desenvolvimento de uma ferramenta para testar automaticamente, o grau de conformidade dessas aplicações, com as normas internacionais de usabilidade e acessibilidade.

A aplicação da ferramenta proposta nas aplicações móveis de um cliente que opera no sector bancário pode significar uma maior satisfação por parte dos seus clientes e por outro lado, expandir os seus produtos a clientes com necessidades especiais, o que poderá refletir-se no aumento da utilização dos seus serviços bancários e consequentemente, o aumento do lucro. No restante do presente capítulo será apresentado o objetivo da ferramenta, a motivação, a abordagem, principais contribuições e a estrutura do documento.

1.2 Objetivo da dissertação

O presente projeto tem como objetivo geral a implementação de uma ferramenta de testes automatizados de acessibilidade para aplicações móveis e a publicação dos resultados na plataforma de gestão de testes Testlink. Como objetivos específicos do projeto foram definidos os seguintes itens:

- Estudar as normas de acessibilidade para Web no geral
- Estudar a sua aplicabilidade nos testes de aplicações móveis;
- Fornecer mecanismos para testes automáticos de acessibilidade de aplicações móveis
- Desenvolver um módulo para execução de testes automáticos a partir do *Testlink*;
- Disponibilizar os resultados dos testes na plataforma Testlink;
- Definir planos de testes de acessibilidade com base nessas normas;
- Executar testes automaticamente.

Como base para o desenvolvimento de um caso de estudo, será utilizada a aplicação do banco Axa Banque, cita em Paris, França.

Espera-se que o produto final satisfaça os objetivos e as necessidades requeridas pela empresa Altran, de forma a merecer a sua implementação no caso de estudo acima referido.

1.3 Motivação

O presente trabalho resulta de uma parceria entre o grupo de investigação RELiable and SEcure Computation (RELEASE) da Universidade da Beira Interior (UBI) e a multinacional francesa Altran. Esta dissertação nasce da necessidade de dotar as aplicações móveis desenvolvidas pela Altran, de uma maior acessibilidade e consequentemente, uma maior usabilidade.

De um modo geral, considera-se que o desenvolvimento de uma dissertação num ambiente empresarial, da qual resulta uma ferramenta que preencha uma lacuna na empresa Altran e resolva um problema concreto de um cliente, é por si só motivante. Por outro lado, tal cenário representa uma oportunidade de pesquisar, analisar e aprofundar os conhecimentos na área da qualidade de software, com ênfase, em testes de aplicações móveis.

1.4 Abordagem

Esta dissertação encontra-se dividida em duas fases principais: a primeira fase, que é a fase de investigação e a segunda, a de desenvolvimento.

A fase de investigação foi subdividida em duas partes: a investigação teórica, onde se estuda os conceitos de engenharia de software, dos testes de software e das normas de acessibilidade; e a investigação técnica, que incidiu nas ferramentas de automação de testes direcionados às aplicações móveis, de forma a perceber o funcionamento dos mesmos, e, melhor definir um

método para a fase de desenvolvimento.

Na fase de desenvolvimento optou-se pela implementação de uma solução que resolvesse o problema apresentado, e também que fosse voltada para o futuro dos testes de acessibilidade de aplicações móveis. Deste modo a ferramenta proposta está capacitada para a integração com outras ferramentas de testes. Com isso, considera-se que foram lançadas as bases para a disponibilização dos testes da acessibilidade na cloud.

Ainda foram implementadas algumas das normas estudadas na fase de investigação teórica que melhor se adequam ao problemas proposto. Paralelamente, foram implementados dois módulos, nomeadamente o *Color Contrast Checker* e o *Brettel Model View*, que quanto a nós, acrescenta valor à ferramenta.

1.5 Estrutura do documento

Este documento está dividido em vários capítulos que mostram o processo desenvolvimento da ferramenta. Neste primeiro capítulo foi descrito a dissertação, as motivações, os objetivos propostos e abordagem.

Capítulo 2 - Estado de Arte dos Testes de Software Neste capítulo apresenta-se uma introdução à engenharia de testes de software, com descrição de conceitos, métodos e técnicas. Este capítulo pretende fornecer uma visão sobre o panorama dos testes de software em geral e os testes aplicações móveis em específico.

Capítulo 3 - Teste de Acessibilidade Digital Neste capítulo apresenta-se um estudo sobre as normas de acessibilidade para conteúdos a Web e sua aplicabilidade aos testes de acessibilidade dos dispositivos móveis.

Capítulo 4 - Análise, arquitetura e metodologia de desenvolvimento

Neste capítulo analisa-se o problema a nível dos métodos de engenharia de software empregues no desenho, desenvolvimento e implementação do sistema.

Capítulo 5 - Implementação da ferramenta ATAT

Neste capítulo descreve-se a implementação, o funcionamento e os resultados da ferramenta.

Capítulo 6 - Conclusão e Trabalho Futuro O último capítulo apresenta as conclusões sobre o trabalho desenvolvido e possíveis trabalhos futuros.

Capítulo 2

Estado de Arte dos Testes de Software

2.1 Introdução

Os testes de software representam a forma mais eficaz de detetar erros e defeitos para futura correção, que quando detetados num ambiente pelos utilizadores, estes erros são suscetíveis de causar não apenas danos económicos, mas também, por em causa a reputação da aplicação em si, e nalguns casos, prejudicar a empresa que a desenvolveu. Tal como os softwares tradicionais, o desenvolvimento de aplicações móveis não é livre de *bugs*, devendo por isso ser desenvolvidas técnicas e ferramentas com o intuito de garantir a qualidade das aplicações móveis.

No presente capítulo apresenta-se os conceitos dos testes de software, e encontra-se dividida em duas secções: testes de software (tradicional) e testes de aplicações móveis. De seguida são apresentados alguns trabalhos relacionados ao tema e uma conclusão do capítulo.

2.2 Testes de Software

A confiabilidade nas funcionalidades de qualquer sistema de software deve ser atestada por mecanismos de testes, que vão permitir detetar a presença de anomalias, que ao serem corrigidas, conferem mais qualidade ao sistema.

Edsger Dijkstra, cientista na área de engenharia de software e computação, afirma que: "Os testes podem apenas mostrar a presença de erros, não a sua ausência"[Ans90].

Na figura 2.1, apresenta-se um modelo de dados de entrada e saída, adaptado do modelo definido por Ian Sommerville [Som10]. Do conjunto de dados de entrada, existe o subconjunto de dados Dt, definido para "provocar" situações de anomalias. Com base no conjunto de dados Dd, é possível detetar comportamentos inesperados e proceder à sua correção.

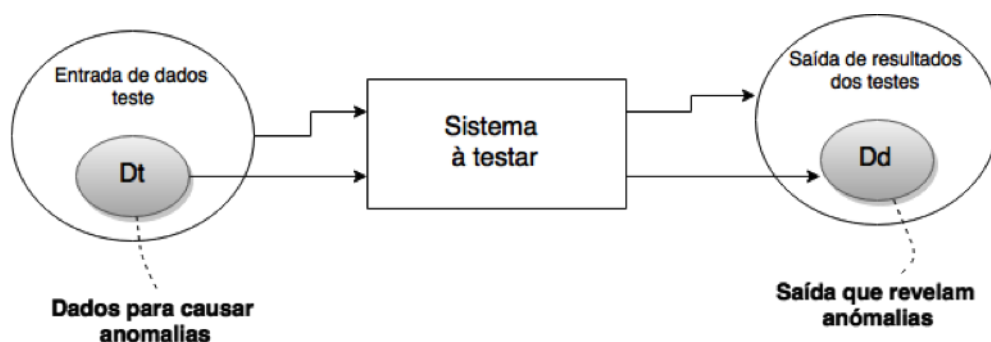


Figura 2.1: Modelo de dados de entrada e saída de um sistema em testes. (adaptado de [Som10])

Devido à impossibilidade de testar o sistema com todas as pré-condições (estado inicial) e combinações de dados de entrada possíveis, torna-se necessário uma certa habilidade, criatividade e conhecimento de técnicas de testes para desenhar um *test suite* (um conjunto de casos de teste) capaz de expor o máximo de erros possíveis [Kan03] [Pat12].

Em casos em que os elementos em teste são subjetivos ou difíceis de quantificar, como é o caso de alguns testes não funcionais (por exemplo, usabilidade e acessibilidade), a tarefa de encontrar dados de entrada também, ela é subjetiva. [Pat12].

2.2.1 A importância dos testes

Os motivos para o lançamento de um produto no mercado, com falhas que possam causar danos e prejuízos económicos são variados. Alguns estão relacionados com estratégias de marketing, aspetos contratuais, e sobretudo com um deficiente planeamento e execução de testes [Tas02]. Os efeitos negativos são materializados na perda de vários milhões de euros, como é o caso do foguetão da *European Space Agency* (ESA), como se exemplifica de seguida:

- Explosão do Araine 5:
 - **Manifestação da falha:** Passados 37 segundos após o início da sequência de ignição do motor principal, deu-se uma perda total de orientação e altitude. Esta perda de informações ocorreu devido a erros de especificação e de projeto no software do sistema de referência inercial.
 - **Causa:** As análises e testes durante o desenvolvimento do programa não incluíram uma análise e testes do sistema de referência inercial ou do sistema de controle do voo, o que poderia ter detetado a potencial falha [LIO96].
 - **Erro:** Falta de tratamento de exceções de um erro de "*overflow*" em que um valor de 64 bits é convertido para 16 bits.
 - **Custo do erro:** Estima-se que os prejuízos ascenderam a meio bilhão de euros.

O relatório elaborado após o acidente, comprovou a importância dos testes de software para garantir o funcionamento do mesmo de acordo com o esperado[LIO96].

2.2.2 Conceitos e Terminologias

A definição de testes de software é variável. Nesta secção apresenta-se alguns dos seus conceitos e terminologias.

Conceito 1: "Teste de um software consiste no processo de análise de um determinado item de um software, de modo a detetar diferenças (defeitos, erros, ou bugs) entre as condições existentes/atuais e as requeridas."(ANSI/IEEE 1059).

Conceito 2: "Testes de software são uma importante componente na área de "*Quality Assurance*", sendo que, as empresas do ramo atualmente gastam cerca de 40% de todos os recursos

Testes automáticos de acessibilidade em aplicações móveis

no processo de testes"[Jov08a].

Conceito 3: "Teste é a atividade que visa avaliar a qualidade de um software, com o objetivo de tomar medidas corretivas. Portanto, o objetivo primordial é encontrar sistematicamente novos erros, com menos esforço e tempo possível"[Jov08a].

Conceito 4: "O teste é geralmente descrito como um conjunto de procedimentos realizados para avaliar algum aspecto de uma determinada parte do software"[Bur02].

Conceito 5: "O teste pode ser descrito como um processo utilizado para descobrir defeitos num software, e para determinar, se o software atingiu um grau específico de qualidade, de acordo com os atributos selecionados."[Bur02].

De seguida expõem-se alguns termos afetos aos testes de software.

- **Erros:** são definidos como uma ação humana que produz resultados incorretos.
- **Falta (fault):** é a causa que levou ao erro. As faltas podem ter origem em qualquer fase do desenvolvimento.
- **Falha (failure):** designa um desvio de comportamento esperado do sistema, que normalmente consiste na manifestação de um erro ocorrido anteriormente.
- **Aplicação sob teste** (Application Under Test (AUT)): trata-se da aplicação que se encontra sob teste.
- **Casos de testes:** Compreende um conjunto de dados de entrada, condições de execução e resultados esperados, desenvolvidos para um objetivo específico como o de exercer um caminho do programa ou para verificar o cumprimento de um requisito específico (IEEE 610 - 1990). Eles representam uma documentação que especifica as entradas, prevê os resultados de saída e um conjunto de condições de execução de um item de teste (IEEE Std 829-1983).
- **Test Suite:** Representa um conjunto de casos de teste.

A qualidade dos testes está diretamente ligada à qualidade dos casos de teste previamente definidos. As boas práticas sugerem que os casos de teste devem ser desenhados de forma a detetar a presença de falhas e não que o programa funciona. Neste sentido, é necessário analisar e estudar técnicas para a criação de casos de teste que permitam encontrar o maior número possível de erros ou falhas[Jov08b].

2.2.3 O processo de testes

O processo de desenvolvimento de um software pode ser descrito como uma série de fases, procedimentos e passos que resultam no produto final: o software. Neste processo, torna-se vital incluir a fase de verificação e validação (V&V) do sistema de software de modo a garantir a satisfação de todos os requisitos especificados ou contratados.

Na Figura 2.2 verifica-se que a atividade de testes é composta por duas partes distintas, porém, complementares: a verificação e a validação.

Barry Boehm, um dos pioneiros de engenharia de software, define as diferenças entre ambos como sendo:

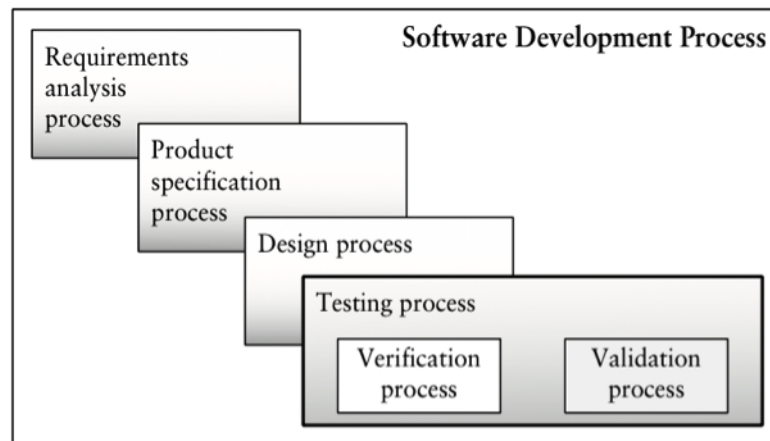


Figura 2.2: Fases de desenvolvimento de um sistema de software. Fonte: (Burnstein2002).

- Validação: *"Are we building the right product ?"*
- Verificação: *"Are we building the product right ?"*

A Validação é o processo que visa avaliar se o software, ou algum dos seus componentes, satisfaz os requisitos especificados anteriormente durante a engenharia de requisitos [Pat12]. Do ponto de vista prático, na validação procura-se encontrar erros que possam existir na fase de implementação, através da execução de casos de teste, previamente desenhados para esse efeito [Bur02]. A validação requer a execução do programa, o que o caracteriza como teste dinâmico.

A Verificação consiste no processo de avaliação de um sistema ou componente, para determinar se o produto numa determinada fase de desenvolvimento satisfaz as condições definidas no início dessa fase [Pat12]. O processo de verificação deve realizar-se, desde a fase de análise de requisitos, onde a análise do projeto e listas de verificação são utilizados como métodos de validação, sendo que, os testes funcionais são executados de modo a garantir a qualidade do produto[Tra99].

Os conceitos 4 e 5 referidos no início desta secção, cobrem as atividades de verificação e validação, bem como todo o domínio dos testes. Mais concretamente tais conceitos cobrem as questões relacionadas com: revisão técnica; planeamento dos testes; rastreamento dos testes; desenho dos casos de teste; testes unitários, de sistema, de aceitação e de usabilidade[Bur02].

2.2.4 Quando e onde começar os testes

Quadri, defende que os testes devem começar nas fases iniciais do projeto [Qua10]. Na Figura 2.3 mostra-se a relação entre o custo da correção de falhas ao longo do tempo/processo de desenvolvimento do software [Qua10]. É possível constatar que encontrar e corrigir um erro na fase de análise de requisitos tem um custo muito menor do que na fase de implementação.

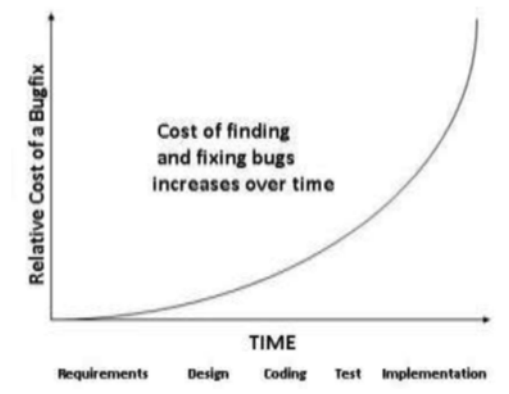


Figura 2.3: Relação entre custo e fase de teste. Adaptado de [Qua10].

O autor considera que o custo da correção do erro após o lançamento do software é de 10 a 100 vezes superior à sua correção na fase de implementação.

2.2.5 Métodos e técnicas de Teste

Os métodos de testes são agrupados essencialmente em dois grupos: métodos de *black box* e métodos *white box*. No entanto, quando utilizados em conjunto, dão origem a um terceiro método: *grey-box testing* [CYZC]. Nesta secção discutimos esses dois métodos e as suas respetivas técnicas.

- Testes de caixa negra (*Black Box Testing*):

Trata-se de um método que consiste na criação de casos de teste, em que, o tester não conhece a estrutura interna do programa [Jov08a], o que significa que este não tem acesso ao código fonte, tornando o sistema opaco para quem o testa. Tendo em conta que o único conhecimento que se tem, é o comportamento do programa, tipicamente, os *testers* fornecem os dados de entrada através do interface de utilizador. Posteriormente, é feita uma análise entre o valor esperado e o verificado efetivamente, sempre que os dois valores não coincidirem [Bur02], será reportado como erro ou defeito, de seguida o caso de teste em questão é devidamente assinalado, para que a equipa de programadores proceda ao "debugging" e à sua correção.

"*Blackbox testing*", é também designado por método de testes funcionais, sendo que os casos de teste têm em conta as especificações do software [Nid12]. Alguns exemplos de técnicas de testes "*blackbox testing*" são apresentadas de seguida. No entanto, tendo em conta que as técnicas não são utilizadas no âmbito deste trabalho não será feita a sua descrição permonorizada, caso o leitor queira obter mais informações, sugerimos a leitura das seguintes referências bibliográficas: [Nid12] [Jov08b] [Wil06] [MRH07].

- *Equivalence Class Partitioning* (ECP): Esta técnica baseia-se na suposição de que os dados de entrada e saída podem ser particionados num número finito de classes (válidos e inválidos). A partição é feita de forma a que, o comportamento do programa seja o mesmo, para todos os casos de testes pertencentes a uma "*equivalence partition*"
- *Decision Tables*

- *Cause-effect Graph*
- *Boundary Value Analysis*

- Testes de caixa branca (*White Box testing*):

Ao contrário do método dos testes de caixa negra, neste caso, a lógica e a estrutura interna do programa é conhecida [Nid12]. Os casos de teste são desenhados a partir do código, o que permite obter, recursos muito eficientes para detetar "*bugs*" (*bug* é uma manifestação de uma falha) antes mesmo de causarem danos maiores [Jov08b].

O objectivo do "*white box testing*" é garantir que diferentes componentes internos que constituem o programa, funcionam corretamente. Tem uma grande aplicabilidade em testes não funcionais, como é o caso de testes de segurança [MO08] [Wil01]. Para isso, os *testers* desenharam casos de teste que exercitam as instruções ou os caminhos possíveis de execução dos programas (*Control Flow* ou *Data Flow*) [Bur02].

Tal como o método anterior, são utilizadas as seguintes técnicas para "*White box testing*":

- *Control Flow Testing*
- *Data Flow Testing*
- *Slice Based Testing*
- *Mutation Testing*

Muitos investigadores consideram que *white-box* e *black-box testing* são complementares entre si. Logo, a forma mais eficaz de testar um software é cobrir os aspetos das especificações e as ações do código [Nid12]. Quando isso ocorre designa-se por *Grey-box testing*. No *Grey-box testing* a interação com o sistema dá-se através do interface do utilizador. É necessário ter algum conhecimento do funcionamento interno do programa ou até mesmo ter revisto código [CYZC].

2.2.6 Níveis de Teste

O processo de testes deve ser dividido em várias fases durante a fase de desenvolvimento. Conforme se pode comprovar na figura 2.4, esse processo pode ser dividido em quatro fases de testes: unitários, integração, sistema e de aceitação.

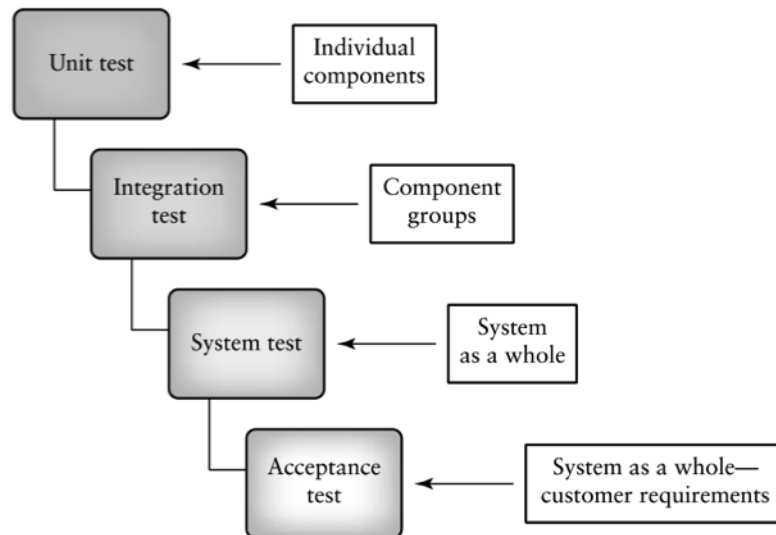


Figura 2.4: Granularidade dos testes de software. Adaptado de [CYZC].

O ciclo de vida dos testes de software inclui a execução dos testes unitários do mais pequeno módulo do software, a junção desses módulos para os testes de integração e a sua combinação com outros subsistemas de modo a que o todo, seja o software final. E por último ocorre a validação e verificação das funcionalidades do sistema num ambiente real, de acordo com as especificações do cliente [CYZC]. Cada fase do ciclo de vida dos testes de software é apresentada a seguir:

- Testes unitários (*Unit testing*) Inspeccionam unitariamente os módulos do programa com o objetivo de encontrar possíveis erros. Os casos de testes são desenhados de acordo com o conhecimento da estrutura interna do programa, pelo que, são utilizadas as técnicas disponibilizadas pelo método "*white-box testing*" [Wil06].

O programador geralmente faz pequenos módulos de código, onde fornece dados de entrada e recebe os de saída da unidade sob teste, posteriormente, esses dados são comparados com os valores esperados para determinar a existência de algum problema na unidade ou módulo [Jen08].

- Testes de integração (*Integration testing*):

Nos testes de integração, os pequenos módulos são integrados em módulos maiores, e estes, integrados num sistema geral em desenvolvimento, de acordo com os requisitos. Estes diferem dos testes unitários na medida em que os módulos deixam de ser testados isoladamente e passam a ser testados em conjunto [Jen08]. Deste modo, testa-se a interação e a coexistência entre módulos.

Ambos os métodos, *white-box* e *black-box testing*, podem ser utilizados neste nível [CYZC]. Os testes de integração contêm vários subníveis de teste que podem ou não ser executados, dependendo do tipo de teste pretendido [Ben05]:

- Testes de compatibilidade (*Compatibility Testing*) são testes utilizados para garantir que a aplicação corre em diferentes configurações do mesmo sistema. Quando se trata de aplicações O programador geralmente faz pedaços de código, onde fornece dados de entrada e recebe os de saída da unidade sob teste, posteriormente, esses

dados são comparados com os valores esperados para determinar a existência de algum problema na unidade ou módulo, consistem em testar o funcionamento em diferentes browsers e diferentes velocidades de conexões.

- Testes de desempenho (*Performance Testing*) são testes que servem para avaliar a escalabilidade quando, por exemplo, o número de utilizadores ou volume de dados aumentam. A abordagem é basicamente recolher os "*timings*" de quando o sistema sob teste, é carregado com poucos utilizadores ("*loads*") e registar seu comportamento gerado pelo aumento gradual de carga.
- Testes de Stress (*Stress Testing*) são testes de performance, mas com um nível e tipo de *loads* superiores. Para determinar o *load* responsável pela falha e a sua causa, as aplicações são testadas nos limites das suas especificações. Uma perda gradual de desempenho, não leva necessariamente, a uma situação catastrófica, no entanto, se um sistema subitamente parar o seu funcionamento, é fundamental saber em que condições isso pode acontecer. Este tipo de teste é muito recomendado para testes de sistemas críticos.
- Testes de carga (*Load Testing*) Este tipo de testes são o oposto do teste de *stress*. Testam a capacidade de uma aplicação funcionar devidamente sob as condições estabelecidas na fase de produção e faz a medição dos tempos de resposta, de modo, a determinar se está dentro dos limites especificados nos requisitos e especificações.

- Testes de sistema (*System Testing*)

Por definição, testes de sistemas são testes conduzidos em sistemas completos e integrados, para avaliar a conformidade do sistema com seus requisitos especificados [Ans90].

Contribuem para a descoberta de inconformidades entre o software e o próprio sistema. O objeto deste teste passa a ser o sistema como um todo, incluindo não apenas software, mas também o ambiente - *hardware* - em que vai correr [CYZC]. Em caso de testes para aplicações móveis, os testes teriam que ser feitos em dispositivos móveis reais, em detrimento de emuladores ou simuladores. Este tipo de testes são muito dispendiosos quando se trata de testar aplicações móveis, quer nativas, quer "*Web mobile apps*".

- Testes de aceitação (*Acceptance Test*)

Testes de aceitação são testes realizados para que o utilizador, cliente ou uma entidade autorizada determine se aceita ou não o sistema ou o componente, com base em critérios de aceitação pré-estabelecidas [IEEE]. Normalmente são testes de caixa negra. Em alguns casos, quando o software é desenvolvido para grandes massas, divide-se em mais dois subníveis: O programador geralmente faz pedaços de código, onde fornece dados de entrada e recebe os de saída da unidade sob teste, posteriormente, esses dados são comparados com os valores esperados para determinar a existência de algum problema na unidade ou módulo [Jen08][Wil06].

2.2.7 Testes manuais e testes automatizados

Os testes de aplicações móveis são tradicionalmente realizados por execução manual dos casos de teste e a verificação visual dos resultados. Contudo é muito dispendioso e requer muito tempo [Muc12]. Por outro lado, é de esperar que os dispositivos móveis recebam dados de entrada de várias fontes e em contextos diferentes o que dificulta a execução de testes manuais [Muc12]. Kim e Na, afirmam que os testes manuais são ineficientes quando se pretende

encontrar erros através da variação das combinações de dados de teste [KNR09], sendo assim, é necessário um novo critério para dar resposta a esse problema.

Henry Muccini, considera os testes automáticos como sendo uma das direções para os testes de aplicações móveis[Muc12].

Testes automáticos são executados por intermédio de um programa informático ou *script*, que é responsável pela comparação entre os resultados atuais e os esperados [Sha14]. Apesar dos benefícios a nível de escalabilidade, rapidez e cobertura, este modo de execução dos testes requer equipas qualificadas e com conhecimento das ferramentas de testes, o que implica algum investimento [Kan]. A automação de testes nem sempre é economicamente viável [Sma15].

2.3 Testes de Aplicações móveis

Wasserman, define aplicação móvel como sendo qualquer software desenvolvido para correr em dispositivos portáteis, tais como, *smartphones* e *tablets* [WF10].

Com os testes de aplicações móveis pretende-se referir a todas as técnicas e métodos de testes tradicionais, aplicados a software desenvolvido especificamente para as plataformas móveis [WF10]. As atividades desta tarefa visam garantir a qualidade das funcionalidades, performance, interoperabilidade, conectividade, segurança, privacidade, mas sobretudo, a usabilidade e acessibilidade.

2.3.1 Tipos de aplicações móveis

As aplicações móveis estão divididas em três categorias, dependendo da sua arquitetura e da tecnologia utilizada no seu desenvolvimento. Perceber os tipos de aplicações e a sua infraestrutura torna-se necessário na escolha do tipo de teste a efetuar, bem como as ferramentas disponíveis. Existem três tipos de aplicações: nativas, *Web apps* e as híbridas.

- Aplicações nativas: são desenvolvidas para uma arquitetura específica e com uma linguagem de programação e *frameworks* pertencentes a esta arquitetura. Normalmente são transferidas das lojas online de aplicações móveis e instaladas nos dispositivos. A sua utilização não está dependente de fatores externos como a conexão à Internet.
- *Webapp*: são aplicações dependentes da ligação à Internet para o seu funcionamento. Os conteúdos estão alojados em servidores Web, portanto, não é necessário a transferência e instalação. O seu desenvolvimento é com base em tecnologias Web, tais como HTML5, CSS and JavaScript.
- Aplicações híbridas: representam uma combinação das aplicações nativas e as *Webapps*. Este tipo de aplicações resultam do desenvolvimento da aplicação com base nas tecnologias Web e incorpora-la em elementos de aplicações nativas.

2.3.2 Diferenças entre testes móveis e testes clássicos

Ao contrário dos testes de software, testes de aplicações móveis requerem técnicas e focos diferentes das aplicações tradicionais. A grande variedade de tecnologias móveis, plataformas, redes e dispositivos, representa um grande desafio no desenvolvimento eficiente de estratégias

de testes. Nesta secção discutimos os desafios que devem ser considerados ao testar aplicações móveis em comparação com as aplicações *desktop*.

Embora muitas das técnicas tradicionais de teste de software possam ser aplicadas aos testes de aplicações móveis, existem numerosos problemas técnicos que são específicos a este tipo de aplicações (discutidos na próxima secção), que diferem dos problemas encontrados nas aplicações para *desktops*. Enquanto nestes, todos os seus componentes já foram amplamente testados a nível da sua compatibilidade [App11], nos dispositivos móveis existe uma maior variedade de componentes, tais como, ecrãs tácteis, teclados virtuais, GPS, *Bluetooth*, Wifi e sinal de rede. Sendo que, a interligação desses módulos, deve ser testado [WF10].

Por outro lado, a diversificação de dispositivos móveis e os seus sistemas operativos com configurações próprias, podem ter comportamentos imprevisíveis a nível de desempenho, segurança e usabilidade de aplicações [IKK14].

2.3.3 Desafios nos testes de aplicações móveis

A portabilidade dos dispositivos móveis torna-os práticos, porem, impõe limitações quando comparado a computadores fixos. De entre essas limitações, e com base no contexto dos testes de acessibilidade, destacamos as seguintes:

- Tamanhos de ecrã: tamanhos muito pequenos são suscetíveis de afetar a usabilidade de aplicações móveis. A apresentação de *Web apps* e aplicações híbridas em dispositivos com ecrã reduzido pode ser pouco *user friendly*, desagradável, com pouca navegabilidade e em muitos casos, ilegível. Em ambos os tipos de aplicações móveis, o tamanho do ecrã pode afetar a usabilidade e a acessibilidade. Portanto, torna-se importante testar a aplicação em diversos tamanhos [LK05] ;
- Resoluções de ecrã: por um lado, baixas resoluções podem degradar a qualidade da informação apresentada, e por outro, diferentes resoluções podem resultar numa representação diferente de mesma informação [LK05];
- Bateria: a fonte de alimentação dos dispositivos móveis é muito limitada. O consumo de energia varia com os recursos utilizados pela aplicação [Bal07]. O comportamento da aplicação em diferentes cenários (pouca bateria) constitui uma importante oportunidade de teste à disponibilidade e acessibilidade [ZA05].
- Reduzido poder de processamento e memória: a capacidade de processamento e memória dos dispositivos móveis, ainda está longe da dos *desktops*.
- Métodos de entrada de dados: a inserção de dados em dispositivos pequenos requer uma certa proficiência, que muitas pessoas podem não ter, ou mesmo, terem perdido com o envelhecimento. Dongsong Zhang e Boonlit Adipat, afirmam em que botões e *labels* pequenos limitam a eficiência na entrada de dados e aumenta a possibilidade de ocorrência de erros. Os teclados virtuais muitas vezes tornam-se um problema em vez da solução [LK05].
- Contexto móvel: a portabilidade do dispositivo faz com que o contexto de utilização seja muito variável e difícil de testar. Dey, Salber e Abowd, descrevem o contexto móvel como sendo "qualquer informação que caracteriza uma situação relacionada com a interação entre utilizadores, aplicações, e o ambiente envolvente" [DAS01]. Segundo Madhushani, as possibilidades são tantas que se torna quase impossível de selecionar uma metodologia de teste de usabilidade e acessibilidade, aplicável a todos os casos [MSMM14].

Testes automáticos de acessibilidade em aplicações móveis

- Quantidade de dispositivos móveis: A diversidade de dispositivos, devido ao modelo de negócio da Google relativamente ao sistema operativo *Android*, é apresentada como sendo um dos maiores desafios de testes de aplicações móveis em geral [Key15].

Outras limitações dos dispositivos móveis, potencialmente, testáveis foram apresentados por Budiu e Travis [Bud15] [Tra13].

2.3.4 Estratégias de testes de aplicações móveis

Um dos objetivos dos testes é a reprodução do ambiente de produção da aplicação. Contudo, devido a limitações de orçamento e de tempo, muitas empresas optam pela estratégia de testes móveis que melhor se adequa aos dois fatores mencionados.

A literatura identifica três estratégias diferentes: testes nos emuladores, testes nos dispositivos reais e uso de serviços de testes na cloud.

1. Testes nos emuladores: é uma estratégia utilizada pelos programadores, muito útil, na fase do desenvolvimento, mas é insuficiente para garantir a qualidade da aplicação [HP14]. A simplicidade, rapidez e preço (normalmente os emuladores já vêm com as ferramentas de desenvolvimento), fazem desta estratégia uma opção de baixos custos[Kam14].

Por outro lado, só os testes nos emuladores são insuficientes e não garantem que o sistema vai correr de igual modo no ambiente de produção. Em outras palavras, não é possível efetuar testes tais como: testes de sistema e performance.

2. Testes em dispositivos reais: representam a estratégia ideal, na medida em que a aplicação é testada num ambiente idêntico ao ambiente de produção. Os testes são mais precisos e não apresentam limitações tais como interoperabilidade, usabilidade e performance, apresentadas pelos testes de sistemas [Key15].

Esta estratégia acarreta um enorme investimento, devido à grande fragmentação do mercado dos smartphones. As boas práticas indicam que se deve testar com 30-40 dispositivos do mercado, devendo 30% destes, ser os modelos mais recentes [HP14].

3. Testes na *cloud* : Consistem em executar em vários dispositivos móveis, e em vários sistemas operativos, as aplicações móveis, onde podem ser testados, atualizados e geridas remotamente, a preços inferiores [IAS12]. Estes modelo de negócio é assente no "*Test as a Service*".

Esta estratégia ultrapassa algumas limitações dos testes de emuladores, elimina a necessidade logística e económica dos testes em dispositivos móveis, mas cria outros problemas, nomeadamente, a perda de controlo sobre a aplicação, a privacidade e sobretudo, em caso de falha na Internet, pode atrasar o processo de testes, e assim, afetar o desenvolvimento da aplicação.

2.4 Ferramenta de gestão de testes Testlink

Testlink é uma ferramenta opensource para a gestão de testes funcionais. Esta plataforma Web, desenvolvida em Hypertext Preprocessor (PHP), permite a equipa responsável pela qualidade do software aglomerar um conjunto de casos de testes em projetos de testes onde é possível a especificação de requisitos, especificação dos testes e bem como a sua execução. A ferramenta

permite para cada teste que se pretenda executar, especificar os seus passos, os respetivos resultados esperados e, após a sua execução, registar o respetivo resultado manualmente. Outra importante característica é a possibilidade de delegar um conjunto de casos de testes para ser executados por um determinado tipo de utilizador.

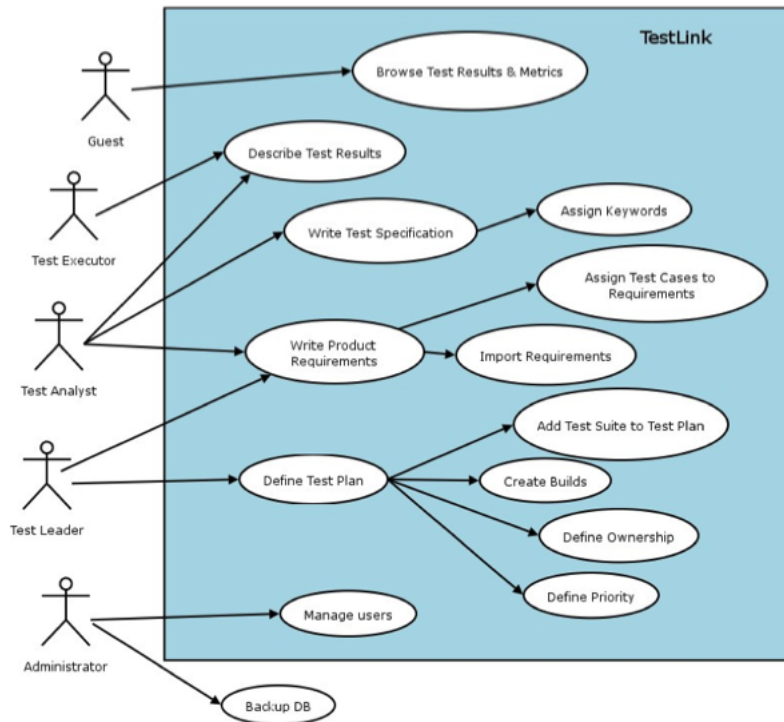


Figura 2.5: Diagrama de casos de uso da Testlink (adaptado de [dSV10])

Na figura 2.5 pode-se ver as diferentes responsabilidades que podem ser delegadas a diferentes tipos de utilizadores na plataforma.

2.5 Trabalhos Relacionados

Em Software Testing, Ron Patton, mostra a importância dos testes ao atribuir 50% do tempo de desenvolvimento de um software para esta fase [Pat12]. Isto inclui testar individualmente todos os componentes do software e posteriormente, testar o sistema como um todo.

A metodologia de desenvolvimento *waterfall*, propõe o início da fase de testes logo após a conclusão da fase de implementação[Roy70]. Contrariando esta metodologia tradicional de desenvolver e testar software, Kent Beck apresenta o Test Driven Development (TDD) [Bec99]. Baseado na metodologia ágil eXtreme Programming (XP), o TDD é apresentado como uma forma eficaz de testar, numa fase embrionária do desenvolvimento do projeto. Essa eficácia foi corroborada pelo Shull, que examina a eficiência do TDD em termos da produtividade, qualidade interna do código e qualidade da entrega do produto([SMT⁺10]).

Na mesma linha, Williams e George, referem que a abordagem do TDD produz código com maior qualidade do que o modelo *waterfall* [GWG03]. Segundo eles, o TDD supera o modelo Waterfall em 18% quando os testes são de caixa negra. Contudo, TDD necessita de mais de 16% do tempo de desenvolvimento que o Waterfall.

Tal como nos testes de software tradicional, há vários métodos de testes de caixa branca e caixa negra aplicados a aplicações móveis, por exemplo, os autores Liu, Yepang, Xu, Chang e Cheung, apresentam uma ferramenta JPF-ANDROID para testes de caixa branca, mais propriamente testes de segurança. Esta ferramenta pode detetar falhas de segurança como *deadlocks* e *race conditions*[LXC12] [MEK⁺12].

A ferramenta MobiTest é apresentada por Bo, Xiang, e Xiaopeng como uma ferramenta de caixa negra para automação de testes em dispositivos móveis [JLG07]. Utilizam os testes baseados em eventos que ocorrem na execução das aplicações. Na nossa perspetiva a ferramenta carece de objetividade. Em contraponto a esse fato, Saswat Anand e os seus colegas, publicaram uma abordagem automatizada para validar o *GUI* através de uma sequência de eventos na aplicação móvel [ANHY12]. Outros trabalhos na área dos testes de GUI, baseados em eventos, foram discutidos em [AFT⁺12] [CYM10].

O teste de conectividade é uma das áreas de testes de aplicações de móveis mais sensíveis. Na maioria dos casos, as aplicações são alimentadas por dados que estão em servidores, por isso é necessário proceder-se à validação da conectividade em diversos contextos, e do tipo de conectividade (Wifi, 3G, 4G e Bluetooth). Satoh, apresenta uma metodologia para testar aplicações com serviços e recursos disponibilizados pela sua conexão atual[Sat04]. Para além da conectividade, a capacidade dos dispositivos móveis de aceder a recursos que não estão disponíveis localmente leva-nos a uma das áreas fulcrais na qualidade das aplicações: a qualidade do serviço (QoS) [Pap13].

Os testes de QoS em aplicações móveis estão diretamente relacionadas com a confiabilidade, tempo de resposta, disponibilidade, escalabilidade e performance. Com foco nesses parâmetros, Rabeb Mizouni, Serhani e Dssouli avaliam o desempenho de Web Services com as tecnologias RESTful e SOAP [MSD⁺11]. Embora, os testes à QoS garantam todos os parâmetros acima referidos, deve-se garantir também, um nível satisfatório de usabilidade e acessibilidade.

A usabilidade corresponde a um dos aspetos mais importantes nas aplicações móveis.

Para construir uma aplicação eficaz, as interfaces de utilizador devem, obedecer às normas e princípios de usabilidade [GM13]. Jacob Nielsen, um dos mais respeitados cientistas na área de usabilidade, define em os cinco princípios de usabilidade: eficiência, satisfação, aprendizagem, memorização e baixa taxa de erros[Nie93].

Kaikkonen e Kallio, realizaram um estudo comparativo entre tipos de testes de usabilidade: testes laboratoriais e testes de campo. Ao contrário do que previram, a maioria dos problemas de usabilidade ocorreram em laboratório e não num ambiente não controlado.

Inserido nos testes de usabilidade, estão os testes de acessibilidade [?].A ISO/IEC Guide 71, define a acessibilidade no contexto das tecnologias de informação, significa providenciar o acesso à informação ao maior número de pessoas, independentemente da sua condição[?]. Derivado à falta de consenso nas definições, Petrie e Kheir defendem que a relação entre a usabilidade e a acessibilidade não é clara [PK07]. Billi, apresenta um método unificado para avaliar a acessibilidade e usabilidade de aplicações móveis (ver figura 2.6) [Bra11].

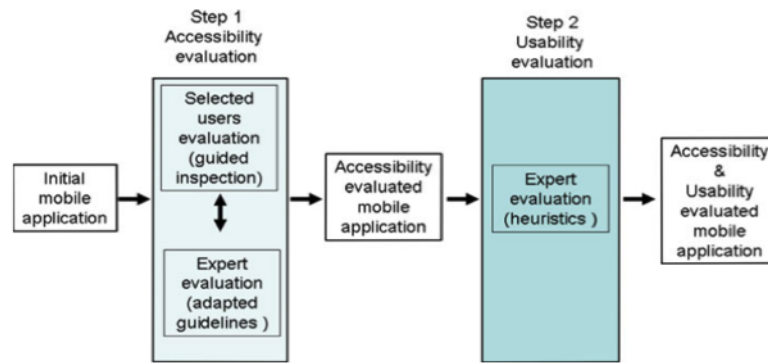


Figura 2.6: Modelo unificado para avaliar acessibilidade e usabilidade Fonte: [BBC⁺10]

Ao testar a acessibilidade primeiro e posteriormente a usabilidade, o autor acredita que ajuda a desenvolver um paradigma universal.

2.6 Conclusão

O processo de testes envolve aspetos técnicos, de gestão, e sobretudo, económicos. As considerações económicas estão diretamente ligadas aos recursos (físicos, humanos e de conhecimento) e o tempo disponível pela equipa de testes [Bur02]. Em muitos casos, os testes não são exequíveis, devido a limitações económicas. No entanto, uma organização ou empresa, deve organizar o processo de testes, de modo a que o produto, seja entregue dentro do orçamento, e também satisfaça os requisitos do cliente.

Os aspetos técnicos referem-se a técnicas, métodos e ferramentas utilizadas, para garantir que o software sob teste contém o mínimo de defeitos e é o mais fiável possível, tendo em conta as condições e restrições em que deve operar [Bur02].

O ato de testar um software é todo um processo, e não um ato isolado, portanto, é necessário uma gestão adequada deste processo. Isto obriga a que as empresas tenham uma política de teste bem definida e documentada. O processo de teste deve englobar mecanismos que facilitem a sua melhoria de forma continua.

No presente capítulo, estudou-se de uma forma geral os testes de software e a sua aplicabilidade nos dispositivos móveis. No entanto, ao contrário dos testes funcionais, os testes de usabilidade e de acessibilidade implicam um enorme conhecimento das normas que as suportam. Neste sentido, no próximo capítulo vamos apresentar o estudo dessas normas e contextualizá-las nos testes de acessibilidade de aplicações móveis.

Capítulo 3

Teste de Acessibilidade Digital

3.1 Introdução

Designa-se por acessível (do latim *accessibile*) tudo aquilo que se pode atingir, alcançar ou obter facilmente, o que é compreensível. (Porto Editora (2001), Dicionário Editora da Língua Portuguesa, 8a Edição.) Assim sendo, o termo acessibilidade digital em aplicações móveis, está associado à disponibilização efetiva de toda a informação para todos os utilizadores, independente da tecnologia ou plataforma utilizada e das capacidades sensoriais, motoras ou funcionais do utilizador.

Para o criador do World Wide Web Consortium (W3C) e actual director da W3C, Tim Berners Lee, *"The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect"*. Com o surgimento do WWW a atenção focou-se na criação de normas e padrões que pudessem servir de diretrizes comuns, de modo a tornar os conteúdos Web mais acessíveis a um maior número de utilizadores[Sha].

Após o surgimento da W3C nasce a *Web Accessibility Initiative* (WAI). WAI é o órgão pertencente ao consórcio W3C encarregue pelo desenvolvimento das estratégias, *guidelines* e recursos que visam tornar a Web, mais acessível a todas as pessoas em geral, tendo estas pouca mobilidade, visão, ou problemas auditivos [Sha].

Shawn Lawton Henry, responsável pela WAI, acredita que os web designers e programadores devem entender a importância da acessibilidade e o quanto uma web acessível aumenta o poder das pessoas com necessidades especiais e da sociedade como um todo[W3C14].

O presente capítulo foca-se no estudo das normas de acessibilidade aplicadas à páginas Web apresentadas pela W3C e da norma *Section 508* dos Estados Unidos da América.

3.2 Acessibilidade num contexto europeu

Na ausência de normas e muitas vezes de leis sobre o tema, existem variadas abordagens no que respeita à acessibilidade das tecnologias de informação, no entanto, é possível dividir-se estas abordagens em três categorias[Hen12]:

- Alguns países requerem que as tecnologias de informação utilizadas e os serviços de informação adquiridos a terceiros, por entidades governamentais, devem ser acessíveis;
- Há países que estabelecem na constituição, que os indivíduos com algum tipo de deficiência têm direito a determinado tipo de informação;
- Os países podem requerer a obrigatoriedade dos produtos e serviços de informação disponíveis no país e de cumprirem com determinados níveis de acessibilidade definidas por organismos internacionais.

Na União Europeia, desde a década de 90, tem existido uma preocupação com o tema, consequentemente foram desenvolvidos inúmeros trabalhos de investigação e desenvolvimento tecnológico, tais como:

- Entre 1991 e 1993, através do projecto - *Technology Initiative for Disabled and Elderly persons* (TIDE)
- Entre 1994 a 1988, através do programa - Telematics Applications Research and Development Programme (TAP);
- Em 1998, com o apoio a WAI da W3C, foram desenvolvidas normas de acessibilidade amplamente utilizadas;
- Em 2000, foi aprovada a iniciativa eEurope: Uma sociedade de informação para todos.

3.3 As Normas de Acessibilidade digital

A W3C/WAI apresenta três guias essenciais para que se possa alcançar a acessibilidade web, sendo que cada guia está diretamente ligada a um componente(ver figura 3.1):

- Conteúdo - Web Content Accessibility Guidelines (WCAG);
- Ferramentas de desenvolvimento de conteúdos para Web - Authoring Tool Accessibility Guidelines (ATAG);
- Utilizador - User Agent Accessibility Guidelines (UAAG);

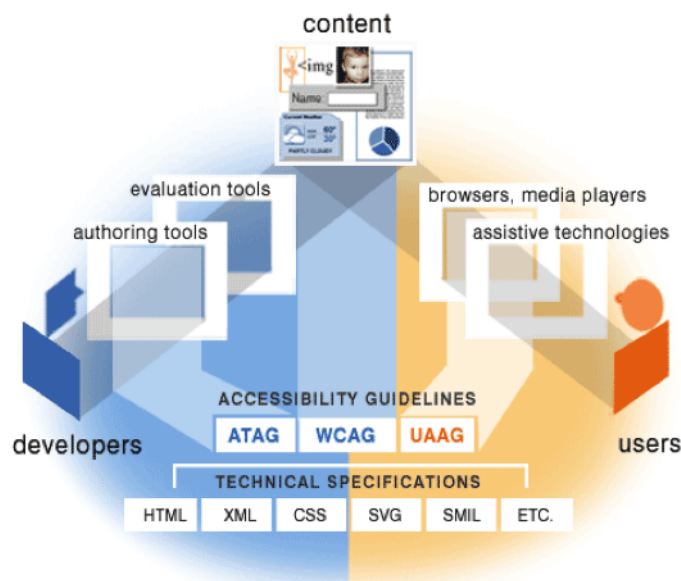


Figura 3.1: As normas de acessibilidade (Fonte: W3C)

Entende-se por conteúdo todo e qualquer elemento textual ou audiovisual, sendo que a norma encarregue de verificar a sua acessibilidade é a WCAG.

As ATAG são normas a ser incorporadas nas ferramentas de desenvolvimento, tais como, Integrated Development Environment (IDE), cujo objetivo é facilitar a integração das normas já nas fases de desenvolvimento.

Enquanto que as normas UAAG, são responsáveis pela acessibilidade nos navegadores, reproduzores multimédia ou qualquer outro componente que faz a renderização de páginas web.

Na W3C acredita-se que a acessibilidade Web, depende do relacionamento entre os três diferentes componentes, e de como, o aperfeiçoamento de componentes específicos, podem melhorar substancialmente as condições de acesso à informação a nível global [Hen12].

No âmbito desta dissertação, debruça-remo-nos apenas sobre as normas para testar a acessibilidade de conteúdos web, que são aplicáveis a testes manuais e automáticos de aplicações móveis.

3.4 Web Content Accessibility Guidelines (WCAG)

As diretrizes de Acessibilidade para Conteúdo Web (WCAG) representam um conjunto vasto de recomendações (previamente validadas com estudos científicos) definidas pela W3C. O objetivo é tornar o conteúdo Web mais acessível.

Ao seguir-se estas diretrizes garante-se também, que o conteúdo Web está em conformidade com regras de usabilidade no geral, beneficiando assim, todos os utilizadores.

"O cumprimento destas diretrizes fará com que o conteúdo se torne acessível a um maior número de pessoas com incapacidades, incluindo cegueira e baixa visão, surdez e baixa audição, dificuldades de aprendizagem, limitações cognitivas, limitações de movimentos, incapacidade de fala, fotossensibilidade bem como as que tenham uma combinação destas limitações". [W3C] Atualmente já foram publicadas a WCAG 1.0 e a WCAG 2.0 (Figura 3.2

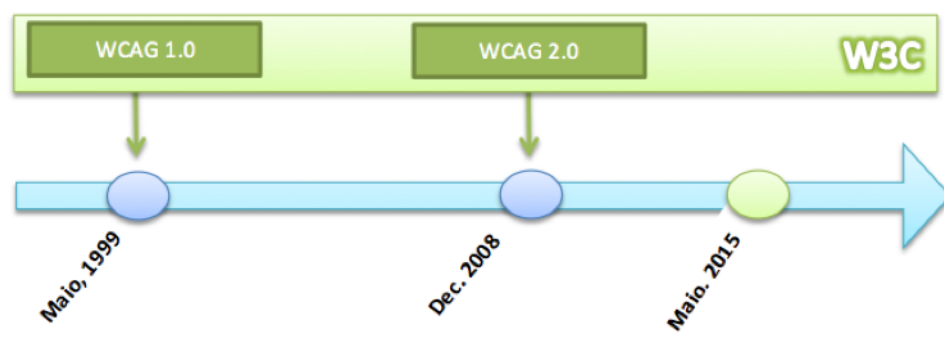


Figura 3.2: Cronologia das versões de WCAG

De seguida apresenta-se as normas referentes a WCAG 1.0 e WCAG 2.0.

3.4.1 WCAG 1.0

Sob o slogan "como tornar o conteúdo Web acessível a pessoas com deficiência", a primeira versão da WCAG, foi lançado em 1999. Composto por catorze guidelines [W3C] [Sik11]:

1. Alternativas equivalentes devem ser fornecidos para o conteúdo sonoro e visual.
2. Informações expressas em cores também devem estar disponíveis e perceptíveis sem cores.
3. As folhas de estilo e formatação devem ser aplicadas corretamente.
4. A linguagem natural dos documentos da Web deve ser declarada.
5. As tabelas devem ser utilizadas para ilustrar informação tabular e não para a estruturação do *layout* de páginas Web.
6. As páginas que aplicam novas tecnologias devem-se transformar harmoniosamente.
7. O utilizador deve conseguir controlar os conteúdos passíveis de mudar com o tempo.

8. Garantir a acessibilidade direta de componentes de user interface embutidos.
9. O desenho do Website deve ser independente do dispositivo.
10. Deve-se explorar soluções internas para acessibilidade.
11. As tecnologias do W3C e as suas diretrizes devem ser aplicadas.
12. A informação deve ser fornecida no contexto e na orientação.
13. A navegação deve ser fácil de entender.
14. Os documentos devem ser claros e simples.

Cada uma dessas diretrizes encontra-se subdividida em Checkpoints, que servem como base para a verificação da conformidade com a WCAG 1.0, na totalidade são 65 Checkpoints, que por sua vez, estão agrupados em três níveis de prioridade:

- **Prioridade 1:** os programadores devem satisfazer estas exigências;
- **Prioridade 2:** os programadores têm de satisfazer estes requisitos, caso contrário, o conteúdo será de difícil acesso para alguns grupos de utilizadores. Segundo Sikos, este nível resolve imensas barreiras de acessibilidade [Sik11];
- **Prioridade 3:** para maximizar a acessibilidade dos seus produtos, os programadores podem satisfazer este nível de prioridade.

3.4.2 WCAG 2.0

Web Content Accessibility Guidelines (WCAG) é desenvolvido e suportado pelo consórcio W3C em colaboração com várias organizações e individualidades espalhadas pelo mundo, sendo que, o seu objetivo é definir diretrizes e padrões para a acessibilidade de conteúdos voltados para a web (<http://www.w3.org/WAI/intro/wcag> , 2015-5-10). O WCAG foi desenhado para ser estável, referenciável e assenta em quatro princípios:

- Percetibilidade;
- Operabilidade;
- "*Understandable*";
- Robusto;

Com base nesses princípios foram apresentados doze guias de acessibilidade. Para cada guia, existem os critérios de sucesso associados (61 no total), que definem a sua testabilidade, bem como, um conjunto de especificações técnicas, que podem ou não ser utilizadas para alcançar a conformidade com as guidelines definidas pela WCAG 2.0 (cf. Figura 3.3).

Testes automáticos de acessibilidade em aplicações móveis

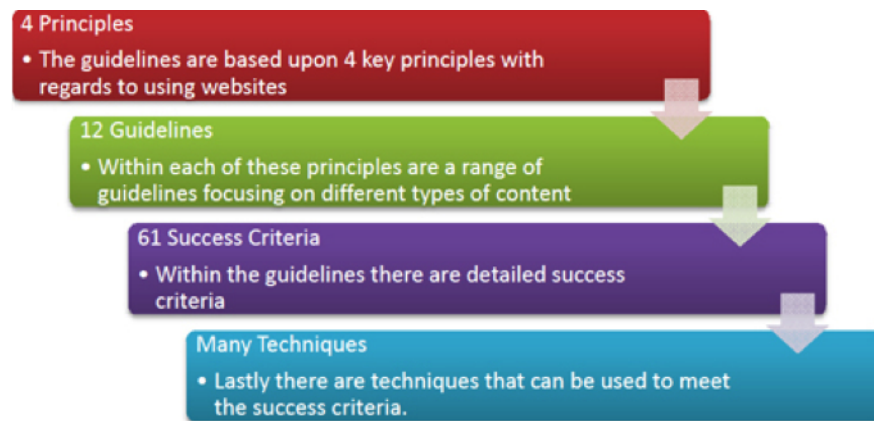


Figura 3.3: A estrutura da WCAG 2.[W3C08]

A WCAG 2.0 define vários níveis de conformidade com as normas de acessibilidade. Esses níveis de acessibilidade são classificadas em:

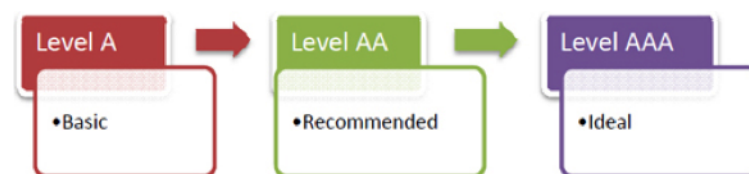


Figura 3.4: Descrição dos níveis de acessibilidade da WCAG 2.0

O nível A corresponde ao escalão mais baixo, o básico, o nível AA ao intermédio e o AAA ao nível máximo. O nível AA garante um nível de acessibilidade adequado a qualquer produto digital, enquanto que o nível A, não permite ter os níveis de acessibilidade desejáveis embora a sua ausência possa implicar a ausência de acessibilidade em geral. O nível AAA é normalmente mais difícil de atingir e representa um acréscimo qualitativo ao nível AA.

A tabela 3.1 Distribuição das guidelines da WCAG 2.0 (Fonte: W3C [W3C08]) mostra a distribuição dos princípios, *guidelines* e os níveis de conformidade.

Tabela 3.1: Distribuição das guidelines da WCAG 2.0 (Fonte: W3C)

Princípios	Guidelines	Level A	Level AA	Level AAA
1. Perceptibilidade	1.1 Text Alternatives	1.1.1		
	1.2 Time-based Media	1.2.1 1.2.3	1.2.4 1.2.5	1.2.6 1.2.9
	1.3 Adaptable	1.3.1 1.3.3		
	1.4 Distinguishable	1.4.1 1.4.2	1.4.3 1.4.5	1.4.6 1.4.9
2. Operable	2.1 Keyboard Accessible	2.1.1 2.1.2		2.1.3
	2.2 Enough Time	2.2.1 2.2.2		2.2.3 2.2.5
	2.3 Seizures	2.3.1		2.3.2
	2.4 Navigable	2.4.1 2.4.4	2.4.5 2.4.7	2.4.8 2.4.10
3. Understandable	3.1 Readable	3.1.1	3.1.2	3.1.3 - 3.1.6
	3.2 Predictable	3.2.1 3.2.2	3.2.3 - 3.2.4	3.2.5
	3.3 Input Assistance	3.3.1 3.3.2	3.3.3 - 3.3.4	3.3.5 -3.3.6
4. Robust	4.1 Compatible	4.1.1 4.1.2		

A documentação disponibilizada pela WCAG, explica como fazer para tornar os conteúdos Web

mais acessíveis para indivíduos com algum tipo de limitação. Portanto, estas guidelines referem-se apenas a páginas Web ou aplicações Web, que incluem informações como o texto, imagem, ou sons, e o código ou linguagens de marcação (ex. HTML, CSS). No entanto a sua aplicação nos testes de aplicações móveis nativas requer um trabalho prévio de análise e adaptação para as diferentes plataformas e tecnologias de desenvolvimento deste tipo de aplicações.

3.5 Mobile Web Best Practices (MWBP)

O MWBP 1.0 designa um conjunto de boas práticas, a serem levados em conta no desenvolvimento de Web apps, de modo a que, os conteúdos sejam facilmente acedidos via dispositivos móveis [BLC10]. Foi desenvolvido para colmatar uma lacuna existente a nível dos padrões no domínio da acessibilidade de Web apps. Estas recomendações são em parte, derivadas do WCAG 2.0 [Rab08].

Assim sendo, potencialmente, é possível fazer o mapeamento ou correspondência entre as duas "guidelines", e consequentemente, tirar partido dos critérios de sucessos definidas na WCAG 2.0 [BLC10].

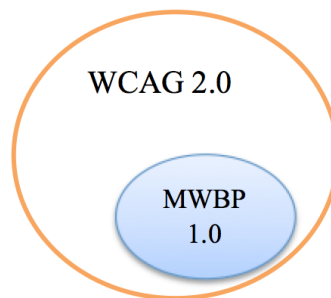


Figura 3.5: Relação entre WCAG 2.0 e MWBP 1.0

O MWBP 1.0 divide-se em cinco temas, contendo cada tema, um conjunto de declarações sobre melhores práticas para o desenvolvimento "mobile web apps":

- Comportamento Geral: define as "*guidelines*" gerais para qualquer dispositivo, independentemente das suas características;
- Navegação e *links*: define como a navegação e os "*links*" devem funcionar de modo a proporcionar maior acessibilidade;
- Conteúdo e *layout* de página: designa como as páginas devem ser desenhadas, e como os conteúdos devem ser criados para dispositivos móveis;
- Definição de páginas: potencia a usabilidade através da exploração das tecnologias Web;
- Interface de entrada de dados: refere como deve levar em conta, os métodos disponíveis para inserção de dados nos dispositivos móveis

A WCAG e o MWBP visam melhorar a interação dos utilizadores cuja deficiência representa uma barreira relativamente ao acesso a páginas Web, ou aplicações (Web app) desenhadas para os dispositivos móveis utilizando tecnologias tais como por exemplo: HTML5, CSS3, XML e JavaScript. Estabeleceu-se uma relação entre o MWBP e a WCAG, tendo em conta o esforço necessário para que o MWBP corresponda a WCAG(ver tabela 3.2) [Cen14] .

Tabela 3.2: Relação entre MWBP e WCAG (adaptado de [Cen14])

MWBP	WCAG
AUTO_REFRESH	3.2.5 Change on Request
FONTS	1.3.1 Info and Relationships.
LINK_TARGET_ID	2.4.9 Link Purpose (Link Only)
NON-TEXT_ALTERNATIVES	1.1.1 Non- text Content
STYLE_SHEETS_USE	1.3.1 Info and Relationships
TAB_ORDER	2.4.3 Focus Order.
USE_OF_COLOR	1.4.1 Use of Color.
BACKGROUND_IMAGE_READABILITY	Possibly covered at level AA by 1.4.3 Contrast (Minimum) and at level AAA 1.4.6 Contrast (Enhanced)
COLOR_CONTRAST	Partially covered at level AA by success criterion 1.4.3 Contrast (Minimum) and at level AAA 1.4.6 Contrast (Enhanced)
LINK_TARGET_ID	Partially covered at level A by 2.4.4 Link Purpose (In Context).
NAVIGATION	partially covered at level AA by 3.2.3 Consistent and at level AAA 2.4.10 Section Headings
NON-TEXT_ALTERNATIVES	partially covered at level AAA by 1.2.7 Full Text Alternative (but covered already at that level by 1.1.1 by Non-text Content)
CONTROL_POSITION	possibly covered at level A by 1.3.1 Info and Relationships
MEASURES	possibly covered at level AA by 1.4.4 Resize text
ACCESS_KEYS	N/A
DEFAULT_INPUT_MODE	N/A
NAVBAR	N/A
TABLES_LAYOUT	N/A
NO_FRAMES	N/A
SCROLLING	N/A
TABLES_NESTED	N/A
PROVIDE_DEFAULTS	N/A
CENTRAL_MEANING	N/A
IMAGE_MAPS	N/A
LIMITED	N/A
OBJECTS_OR_SCRIPT	N/A

A WCAG 2.0 e o MWBP têm abordagens ligeiramente diferentes, na medida em que, enquanto a WCAG 2.0 estabelece técnicas concretas para se testar os critérios de sucesso, a MWBP não o faz.

A WCAG e MWBP 1.0 são "guidelines" que visam melhorar a acessibilidade dos conteúdos das páginas e aplicações Web, a Section 508 tem maior abrangência. Na próxima secção expomos de forma mais detalhada essa norma.

3.6 Section 508

As *guidelines* definidas na Section 508 fazem parte de uma extensa legislação norte-americana, *The Rehabilitation Act of 1973* [Reh]. Essa legislação inclui um conjunto leis que visam que pessoas com limitações físicas, possam mais facilmente, ultrapassar tais barreiras, tanto no dia-a-dia, como, no uso das novas tecnologias de informação. Para o contexto do nosso estudo,

existem nesta legislação, duas secções com impacto na acessibilidade digital: Section 508 e Section 504.

A Section 508 fornece-nos um *blueprint* do que é descrito pela Section 504, assim sendo, a Section 504 representa o contexto da lei e a Section 508 é a direcção a seguir, para dotar um *Website* ou aplicação de uma maior acessibilidade[Web] . Os padrões técnicos apresentadas na Section 508 estão distribuídos por quatro sub-secções:

1. Geral:

- Propósito (1194.1);
- Aplicação (1194.2);
- Exceções gerais (1194.3);
- Definições (1194.4);
- Equivalência (1194.5).

2. Especificações técnicas:

- Aplicação em Software e sistemas operativos (1194.21);
- Informação disponibilizada na Web e aplicações (1194.22);
- Produtos de telecomunicações (1194.23);
- Produtos Multimédia e vídeos (1194.24);
- Produtos fechados (1194.25);
- Computadores portáteis e fixos (1194.26).

3. Critérios de desempenho funcional (1194.31);

4. Informação, documentação e suporte (1194.41).

Durante o nosso estudo, iremo-nos focar nos pontos 2 e 3, Especificações técnicas e Critérios de desempenho funcional, respectivamente. As Especificações técnicas estão organizadas por parágrafos, e identificados por uma letra do alfabeto. Para o caso da *guideline* Aplicação em Software e sistemas operativos (1194.21), existem um total de 12 parágrafos, portanto, de (a) a (i). Por exemplo, a referência 1194.21(a) diz que: *"When software is designed to run on a system that has a keyboard, product functions shall be executable from a keyboard where the function itself or the result of performing a function can be discerned textually"*[Reh98]. Ao contrário das outras *guidelines* já mencionados, a Section 508 faz parte de uma legislação, logo, o seu uso é obrigatório em qualquer instituição pública nos Estados Unidos da América, e quaisquer instituições privadas que recebam fundos governamentais[Reh98].

3.7 Relação entre Section 508 e as diretrizes da W3C

A secção (1194.22) da Section 508 é destinada à acessibilidade em páginas web e aplicações (Informação disponibilizada na Web e aplicações) e foi inspirada nas *Checkpoints* definidas na WCAG 1.0 [Reh98]. Portanto, nesse sentido, pode-se dizer que a *guideline* 1194.22 e as *checkpoints* da WCAG 1.0 são consistentes, a tabela 3.3 demonstra esta correspondência:

Tabela 3.3: Relação entre Section 508 e a WCAG 1.0

Section 508	WCAG 1.0
Paragrafo 1194.22	Checkpoints
(a)	1.1
(b)	1.4
(c)	2.1
(d)	6.1
(e)	1.2
(f)	9.1
(g)	5.1
(h)	5.2
(i)	12.1
(j)	7.1
(k)	11.4

Ao estabelecer este paralelismo, torna-se credível adaptar a Section 508 às nossas necessidades, na tentativa de colmatar a ausência na bibliografia estudada, de normas específicas, para os testes de acessibilidade em aplicações móveis.

Por outro lado, pensamos que em quanto mais normas os testes se basearem, maior será a credibilidade da ferramenta de testes automático de acessibilidade, que pretendemos desenvolver.

No que respeita às vantagens e desvantagens da Section 508 e da WCAG 2.0, a documentação disponibilizada pelo WAI/W3C, das diretrizes WCAG 2.0 é vista por muitos programadores, como sendo confusa e ambígua. Casos há, em que, as *guidelines* conjuntamente com as técnicas propostas para satisfazer os critério de sucesso, são demasiado extensos e de difícil percepção [Sik11]. Em contraponto, a Section 508 apresenta menos informação textual e de forma mais sucinta e clara. No entanto, apresenta a desvantagem de não estabelecer concretamente os critérios de sucesso, bem como, a respetiva técnica de teste, para cada um dos parágrafos [JT].

3.8 O porquê da acessibilidade nas aplicações

Os problemas de acessibilidade advêm de vários fatores, uns ligados a problemas de deficiência, e outros diretamente ligados à perda de habilidades físico-motoras, ou até, mesmo cognitivas [DBM13] [Gab10]. Com base nisso, categorizou-se estes problemas em quatro categorias diferentes: problemas de ordem visual, físico-motoras, audição e cognitivo (ver tabela 3.4) [Kon96].

Tabela 3.4: O porque da acessibilidade

Deficiência	Exemplo de problemas
Sem visão	Não consegue utilizar o rato;
	Não pode ver o ecrã;
Pouca Visão	Dificuldade com áudio e vídeo
	Dificuldade com contraste
Audição	Não pode ouvir áudio ou vídeo
	Não consegue ouvir alertas ou alarmes
Mobilidade	Limitações no uso da mão
	Tremores
	Limitações no raio de ação
	Pouca destreza
Cognitivo	Dificuldades de leitura
	Dificuldades de compreensão
	Dificuldades de escrita

Muitos desses problemas correlacionam a idade com a deficiência. A perda de visão e audição com o avançar da idade é um problema que afeta boa parte da população mundial [MM12].

3.9 A abordagem para Automatizáveis

Durante o nosso estudo, deparámo-nos com um défice de *guidelines* relativas especificamente a testes de acessibilidade, nas aplicações móveis nativas.

Com base nisso, fizemos a correlação entre a WCAG 2.0 e Section 508. O resultado obtido (ver tabela 3.5), está organizada da seguinte forma:

- *Element*: identifica o elemento a testar(p. Ex. Texto, áudio, legenda, imagem);
- *Guidelines*: Representa as normas utilizadas;
- *Test Type*: Corresponde ao tipo de teste pretendido ou possível de fazer(p. Ex. Automático, Semiautomático, Manual).

Tabela 3.5: Relação entre WCAG 2.0 e Section508

Element	Guidelines		Test Type
	Section 508	WCAG 2.0	
Texto	1194.22 (a)	Non-text Content:	Automático
Imagem	1194.22 (e) 1194.22 (f)	N/A	
Audio	1194.22 (b)	1.2.1 Audio-only and Video-only (Prerecorded)	Automático
Legendas	1194.22 (b)	1.2.2 Captions (Prerecorded)	N/A
Audio and Video	1194.22 (b)	1.2.1 Audio-only and Video-only (Prerecorded)	Semi-automatico
Informação e sua estrutura	N/A	1.3.1 Info and Relationships	Automático
Formulários	1194.22 (n)	1.3.1 Info and Relationships (Level A)	Automático
CSS	1194.22 (d).	1.3.1 Info and Relationships (Level A)	
Color	1194.22 (c) Web	1.4.1, Use of Color, (Level A)	
Audio control		1.4.2, Audio Control (Level A).	
keyboard		2.1.1, Keyboard (Level A)	Automático
	1194.22 (l)	2.1.1, Keyboard, (Level A)	
Navigation	1194.22 (o)	2.4.1, Bypass Blocks	
	1194.22 (i)		Automático
		2.4.3, Focus Order, (Level A)	
		3.2.1, On Focus (Level A)	Automático
		3.3.2, (Level A)	

Na secção seguinte aborda-se os tipos de testes de acessibilidade abordados.

3.10 Tipos de Testes de acessibilidade

Nesta secção, descrevemos as três formas de execução dos testes de acessibilidade: testes automáticos, semiautomáticos e manuais.

Automáticos: são todos os testes executados por um software ou script. A decisão final (Pass or Fail) é tomada sem intervenção humana.

Exemplo: Para testar a acessibilidade para elementos não textuais, em aplicações Web, segundo a Section 508, 1194.22a e com a correspondência na WCAG 2.0 1.1.1, é necessário ver a existência de uma descrição (p. ex: "alt" ou "longdesc") no DOM (Document Object Model) correspondente. Caso não exista, o teste falha.

O mesmo teste no Android seria realizado com análise da existência ou não, do atributo *android:contentDescription*, nos elementos não textuais [Goo].

No iOS, este teste seria feito com recurso ao *Application Programming Interface (API)*, *UIAccessibility* [Inc]. Neste caso, a não existência do atributo Label faz com que o teste falhe. Caso contrário, o teste passa.

Semiautomáticos: são todos os testes que necessitam de supervisão humana na decisão final. No entanto, parte deste teste, é executado com o auxílio de um software ou *script*.

Exemplo: No elemento Áudio e Vídeo da Section 508 - 1194.22 (b) diz que : *"Equivalent alternatives for any multimedia presentation shall be synchronized with the presentation."*

Mas em relação à sincronização referida em 1194.22(b), pensamos que deve ser testada pelos humanos. Pois, a automatização dessa tarefa, significaria o desenvolvimento de duas ferramentas: *Text-to-Speech* e o correspondente *Speech-to-text* [AZAS10] [SFKS03]. Julgamos que este ponto, foge do âmbito e não constitui o objetivo principal desta dissertação.

Manuais: Neste caso, não há qualquer intervenção de qualquer software no teste. A verificação e decisão final é feita por um humano. Nem sempre, a automação do teste é a melhor solução. Há vários aspetos a ter em consideração antes de definir o tipo de testes a efetuar [Mar00].

3.11 Conclusão

Com a proliferação das tecnologias de informação na nossa sociedade, mais precisamente, dos dispositivos móveis, torna-se fundamental aliar o aumento da procura à qualidade de utilização, independentemente das características de quem acede à informação. Uma das formas de garantir a universalidade da informação e combater a infoexclusão é através de testes de usabilidade e acessibilidade. No entanto, estes tipos de testes requerem um conhecimento prévio das normas que lhes conferem credibilidade e sustentabilidade. No presente capítulo, foi apresentado um estudo sobre essas mesmas normas e a sua aplicabilidade nos dispositivos móveis. Constatamos que nessa área, ainda existe um défice de normas e ferramentas pensadas exclusivamente para os testes de acessibilidade de aplicações móveis. Ao estabelecer paralelismos entre as normas acima mencionadas, foi-nos possível vislumbrar um ponto de partida para o desenvolvimento do protótipo de uma ferramenta de testes de acessibilidade automatizados, suportada pelas convenções internacionais. Neste sentido, vamos canalizar os nossos esforços nos testes das aplicações moveis com base nos **Mobile Web Best Practices**, que são suportadas pelas normas WCAG 2.0, WCAG 1.0 e em parte , pela Section 508. No próximo capítulo iremos descrever as fases de engenharia de software necessárias para estudo, desenho e implementação do protótipo.

Capítulo 4

Análise, arquitetura e metodologia de desenvolvimento

4.1 Introdução

Num contexto tecnológico, existe cada vez maior preocupação com aspetos de qualidade no geral, e da qualidade das aplicações móveis em particular, torna-se imperativo garantir a qualidade das aplicações antes de sair para mercado. Para isso é necessário definir uma abordagem que integra métodos, procedimentos e ferramentas para o desenvolvimento de software. Esta abordagem é muitas vezes designada de metodologia de desenvolvimento.

Este capítulo trata dos processos de engenharia de software empregues na construção da ferramenta proposta, denominada de *Accessibility Tests Automation Tool*.

As metodologias de desenvolvimento de software têm como objetivo, colocar ordem numa atividade inerentemente caótica, como é o desenvolvimento de produtos de software [Pre05]. Portanto, neste capítulo será abordada a metodologia ICONIX aplicada ao processo de desenvolvimento da ferramenta proposta, suas fases e os artefactos produzidos. Aborda-se também, a linguagem de modelação *Unified Model Language (UML)* e a sua utilização no processo de desenvolvimento ICONIX.

Neste capítulo aborda-se as metodologias de desenvolvimento, a análise de requisitos, a análise e desenho preliminar e o desenho detalhado.

4.2 Metodologia de desenvolvimento ICONIX

Para o desenvolvimento do protótipo utilizou-se a metodologia ICONIX que foi introduzida no ramo em 1993 por Doug Rseberg [RS07]. ICONIX consiste no desenvolvimento incremental e em paralelo dos artefactos que retratam os modelos dinâmico e estático de um sistema, privilegiando a “rastreadibilidade” (*Traceability*) e a robustez. A figura 4.1 ilustra a divisão destas duas visões e os seus respetivos modelos em UML. Apesar dos diagramas de robustez não fazerem parte das especificações UML, estes desempenham um papel importante na definição do conceito de rastreadibilidade presente nesta metodologia, e que permite através da análise de robustez, efetuar a transição entre a fase de análise e a fase de desenho [RS07].

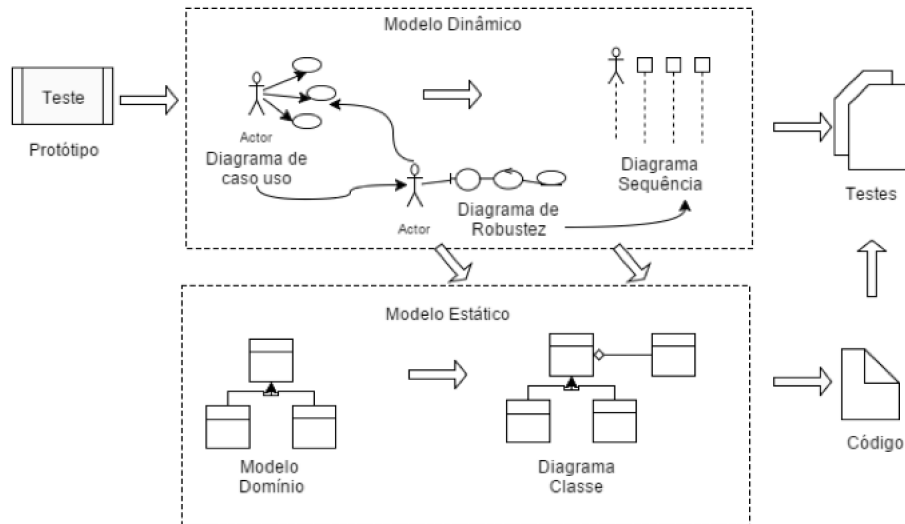


Figura 4.1: Diagramas da visão estática e dinâmica do ICONIX (adaptado de [dSV10])

O processo definido pela metodologia ICONIX segue uma abordagem essencialmente prática, que promove a modelação de um sistema segundo o paradigma orientado a objetos, seguindo um princípio de que se deve modelar e desenhar incrementalmente, o que o torna uma metodologia iterativo e incremental. Em comparação com outras metodologias existentes, ICONIX é simples e prática sem a complexidade do Rational Unified Process (RUP) e com a simplicidade de, ao mesmo tempo, manter a capacidade de análise e de desenho de grandes sistemas de software [dSV10]. De referir que o estudo aprofundado das metodologias de desenvolvimento de software não se enquadra no âmbito desta dissertação.

O ICONIX engloba quatro tarefas principais:

1. Análise de requisitos
2. Análise e desenho preliminar
3. Desenho detalhado
4. Implementação

A figura 4.2 apresenta uma visão geral da metodologia ICONIX e a sequência de tarefas bem como os modelos UML, a serem produzidos em cada tarefa o que comprova importância dos modelos de UML nesta metodologia.

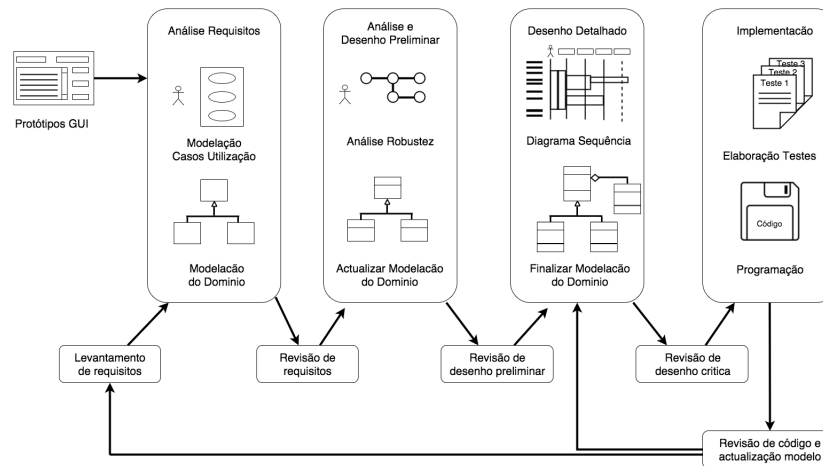


Figura 4.2: Visão geral da metodologia ICONIX (Fonte: [dSV10])

Nas secções seguintes descrevem-se as tarefas incluídas no processo de desenvolvimento com base na metodologia ICONIX.

4.3 Análise de requisitos

A primeira atividade do processo ICONIX é a análise de requisitos. Após a sua conclusão, deve-se modelar o sistema com um modelo de domínio e a modelação dos casos de uso.

Os requisitos de um sistema de software são as descrições do que sistema deve fazer, os serviços que ele oferece e as suas restrições operacionais. Estes requisitos devem refletir as necessidades dos clientes por um sistema que tenha um propósito concreto. Todo este processo de pesquisa, análise, documentação e validação desses serviços e as suas restrições é chamado de engenharia de requisitos (ER)[Som10].

A origem da maioria dos sistemas de software está relacionada com as necessidades de determinados clientes. Sendo que, normalmente o sistema de software é desenvolvido por uma equipa de programadores. Posteriormente, o sistema será utilizado pelos utilizadores finais [Jal08].

Para que se possa transformar o problema proposto em um sistema de software que satisfaça todas as três partes (o cliente, utilizador e o programador), é necessário que se faça uma análise do problema, levantamento dos requisitos e das especificações. É exatamente nesse ponto que a ER se torna fundamental para todas as fases subsequentes de desenvolvimento do sistema. Um erro na fase de ER poderá significar um erro na fase de implementação. Portanto, a qualidade da ER tem uma implicação direta na qualidade da solução final e no custo total do sistema [Boe84].

De forma geral, a ER de sistemas de software consiste num conjunto de atividades desencadeadas, para definir objetivamente, o propósito a que se destina o software, identificando e documentando as necessidades de todas as partes interessadas.

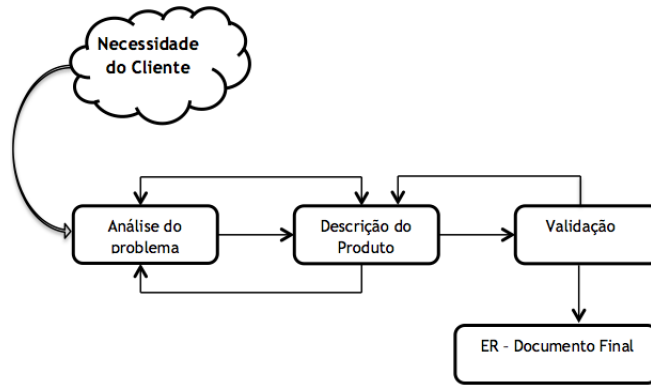


Figura 4.3: Processo de Engenharia de requisitos (adaptado de [Jal08])

A figura 4.3 ilustra o processo de ER. Conforme se pode verificar na figura, após o levantamento das necessidades do cliente, segue-se a atividade de análise do problema, que dá origem a um conjunto de especificações (descrição do produto). Frequentemente, é preciso voltar da atividade de descrição do produto para o de análise do problema, sobretudo quando há dependências entre especificações. Quando as especificações estão concluídas, segue-se a atividade de validação das mesmas. Em caso de algum erro, a especificação volta para ser analisada. O resultado final é um documento com o conjunto de especificações validadas. Esse documento é transversal ao processo de desenvolvimento, incluindo nos testes de aceitação por parte do cliente.

4.3.1 Classificação dos Requisitos

A análise dos requisitos fornece a especificação do que deve ser implementado no software. Os requisitos normalmente são classificados em funcionais e não-funcionais, dependendo se eles descrevem o comportamento que o sistema oferece ao utilizador final ou as condições e possíveis restrições em que devem operar, respetivamente.

Requisitos Funcionais: representam as especificações de serviços e funcionalidades que o sistema deve fornecer, bem como o comportamento do sistema perante certas entradas. Os requisitos funcionais são suportados pelos requisitos não funcionais, que lhes garantem requisitos mínimos de qualidade para o seu normal funcionamento[PDKK02][Lat14].

Requisitos não funcionais: referem-se a restrições e propriedades comportamentais do sistema, tais como, a eficiência, portabilidade, segurança, confiabilidade, usabilidade e disponibilidade [PDKK02][Lat14].

Com o intuito de alinhar a nossa estratégia de ER com as boas práticas de gestão de requisitos seguidas pela Altran, utilizamos o critério de objetivos SMART para a avaliação dos requisitos. Os objectivos SMART (*Specific, Measurable, Achievable, Relevant, and Time-based*) são uma forma de gestão orientada para os objetivos, onde se espera que os mesmos sejam definidos de forma específica, mensurável, atingível, relevante e com um tempo associado[Bog05].

A abordagem para a definição do que chamamos de requisitos SMART, segue os seguintes critérios:

- *Specific* (Específico): É claro para qualquer pessoa com conhecimentos básicos do projeto.
- *Measurable* (Mensurável): Saber se um requisito pode ser satisfeito, como será e quando o será.

Testes automáticos de acessibilidade em aplicações móveis

- *Achievable* (Atingível): Saber se existem recursos para tornar o requisito possível.
- *Relevant* (Relevante): Saber se o requisito é relevante ou não no contexto atual do projeto.
- *Time-Based* (Temporização): Quantificar o tempo (em horas ou dias) da duração da implementação do requisito.

Na tabela 4.1 podemos ver um exemplo de como foram definidos os requisitos para o sistema. Uma outra tabela mais extensa com os requisitos SMART poderá ser consultada no Anexo YY.

Tabela 4.1: Requisitos funcionais

Requisitos Funcionais							
ID	Requisito	S	M	A	R	T	Status
RF01	Execução de teste de acessibilidade das Mobile Web Best Practices 1.0	SIM	SIM	SIM	SIM	5	100%
RF02	Escolher normas	SIM	SIM	SIM	SIM	5	100%
RF03	Criação de projectos de Testes	SIM	SIM	SIM	SIM	1	100%
RF04	Criação de Planos de testes de acessibilidade	SIM	SIM	SIM	SIM	1	100%
RF05	Criação de builds	SIM	SIM	SIM	SIM	1	100%
RF06	Criação de Test Suit	SIM	SIM	SIM	SIM	1	100%
RF07	Criação de casos de testes	SIM	SIM	SIM	SIM	1	100%
RF08	Poder definir o tipo de execução dos casos de testes (automatico,semi-automatico, manual)	SIM	SIM	SIM	SIM	1	100%
RF10	Disponibilização dos resultados dos testes em XMLe/ou JSON	SIM	SIM	SIM	SIM	4	100%
RF11	Fazer o parser do XML para POJO	SIM	SIM	SIM	SIM	5	100%
RF12	Fazer o parser do POJO para JSON	SIM	SIM	SIM	SIM	1	100%
RF13	Módulo de comunicação com Testlink - Servidor de testes (Half-duplex)	SIM	SIM	SIM	SIM	5	100%
RF14	Módulo de comunicação Servidor de testes - Testlink	SIM	SIM	SIM	SIM	4	100%
RF15	Disponibilização de Webservices para todos os testes automatizaveis	SIM	SIM	SIM	SIM	3	100%
RF16	Receber requisições via Aplicação Web	SIM	SIM	SIM	SIM	1	100%
RF17	Definir os passos dos testes para os testest manuais	SIM	SIM	SIM	SIM	1	90%
RF18	Analizar a acessibilidade do background e foreground	SIM	SIM	SIM	SIM	5	100%
RF19	Implementar algoritmos W3C, para cálculo do contraste	SIM	SIM	SIM	SIM	2	100%
RF20	Implementar algoritmo W3C, para cálculo do luminosidade	SIM	SIM	SIM	SIM	2	100%
RF21	Implementar o Algoritmo/Model de Brettell aplicado aos testes	SIM	SIM	SIM	SIM	4	90%

O campo ID: Um requisito funcional deve conter um identificador único e uma breve descrição. Estas informações simplificam o rastreamento das funcionalidades implementadas e as que faltam implementar. Por outro lado, são também utilizados nos testes de aceitação do sistema. Para distinguir o ID dos requisitos funcionais dos não funcionais, usamos a anotação, RF00 e RNF00, respetivamente, em que a parte numérica é incrementada ao criarmos um novo requisito.

Requisito: descrição simplificada do requisito. **SMART:** podem tomar dois valores (Sim ou Não). **Status :** a percentagem que quantifica o progresso de um requisito.

Após a descrição dos requisitos, ICONIX sugere a modelação do sistema com base em casos de uso, identificando os seus atores. De seguida apresenta-se o diagrama de caso de uso.

4.3.2 Diagrama de casos de uso

Os diagramas de caso de uso descrevem a relação entre os atores (*test manager* e *tester*) e a utilização do sistema. O processo de identificar os casos de uso e os respetivos atores, constitui uma das tarefas basilares na engenharia de software, nomeadamente, na especificação de requisitos e/ou de modelação de processos de negócio.

Para uma melhor perceção do funcionamento de todo o sistema, elaborou-se um diagrama de casos de uso (ver figura 4.4), que representa uma visão geral da interação dos atores com o sistema.



Figura 4.4: Visão geral das funcionalidades do sistema

Na secção seguinte pode-se consultar um possível modelo de domínio para suportar a solução a proposta de acordo com os requisitos.

4.3.3 Modelo de domínio

Este modelo pertencente ao modelo estática e consiste em identificar as entidades e as suas possíveis relações, de modo a conseguir compreender os principais conceitos do sistema. Como resultado obtém-se os diagramas de classes de alto nível ilustrado na figura 4.5.

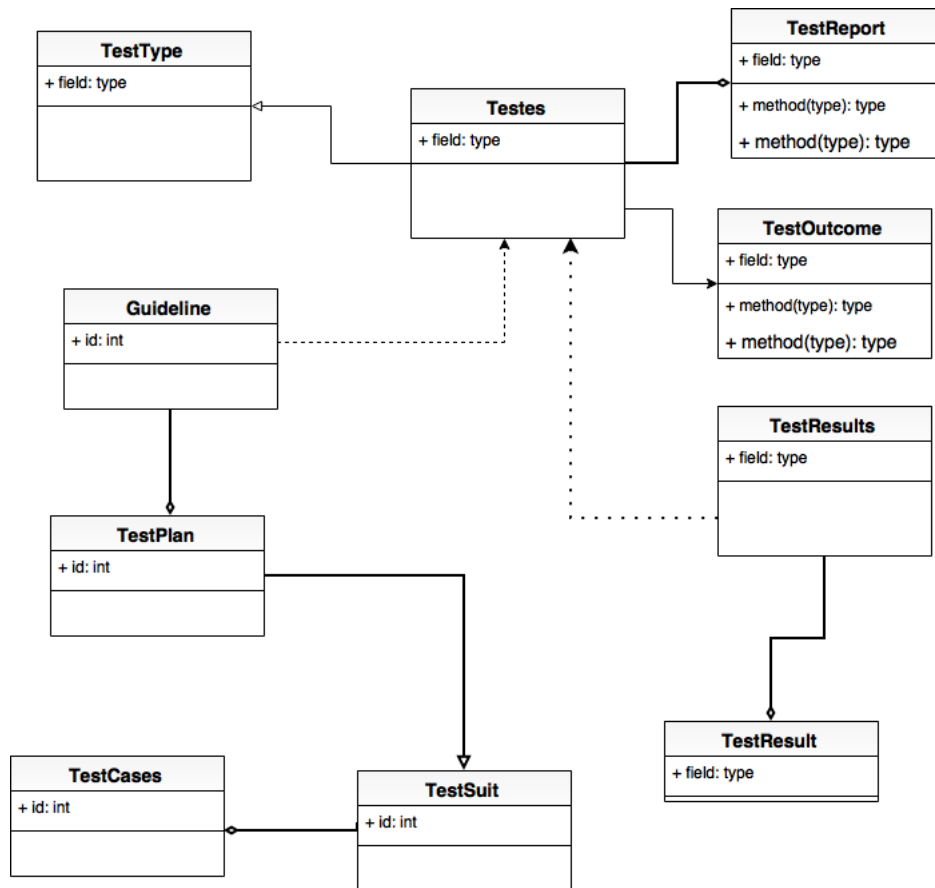


Figura 4.5: Modelo de domínio

É importante referir que o processo de elaboração do modelo de domínio é iterativo e incremental tal como defendido pelo ICONIX, o que significa uma abertura para alterações nas ligações e possibilidade de acrescentar classes ao modelo até à produção final do modelo estático (ver 4.1).

Na secção seguinte será apresentada a segunda tarefa conforme definida acima.

4.4 Análise e desenho preliminar

A primeira atividade da análise e desenho preliminar é fornecer uma transcrição entre a análise e o desenho do sistema. Para tal, elabora-se uma descrição textual para cada cenário correspondente a cada caso de uso, onde se contempla as pré-condições, os fluxos principais, os alternativos e as exceções. A análise de robustez é a segunda atividade desta fase e visa a eliminação de possíveis inconsistências e ambiguidades entre os diagramas de casos de uso e as descrições textuais[dSV10]. Nesta secção apresenta-se uma descrição dos casos de uso. Descrever todos os casos de uso resultaria num trabalho extenso, portanto, iremos apresentar apenas os casos de uso que considera-se fulcrais para o sistema, tais como: “criar projeto de testes”, “criar plano de testes” e “executar testes”. Adota-se a seguinte nomenclatura para a identificação de onde poderá ocorrer o desvio do fluxo principal: FP_N. Em que FP representa o fluxo principal e N o número do passo do mesmo. A criação de um plano de testes de acessibilidade com todas as informações necessárias para a sua execução está retratada na descrição do referido caso de uso, que se encontra na tabela 4.2.

Testes automáticos de acessibilidade em aplicações móveis

Tabela 4.2: Descrição do caso de uso “Criar Plano de testes” de acessibilidade

Nome	Criar Plano de testes
Pré-condições	Ter efectuado o Login no sistema
Objetivo	Criação de um plano de testes de acessibilidade com os parâmetros necessários para a execução dos testes. Esses parâmetros são URL da WebApps, seleccionar o projeto, o plano de teste e o conjunto de MWBP pretendidos.
Fluxo principal	<ol style="list-style-type: none"> 1. Aceder a ferramenta de testes de acessibilidade (NOME), 2. Navegar para a aba Test Plan 3. Fornecer a URL da Webapp 4. Seleccionar o projeto pretendido 5. Seleccionar o plano de testes pretendido 6. Seleccionar um ou mais MWBP 7. Submeter a criação do projeto através do botão “Submit”
Fluxo alternativo	1. [FP_1] O ator clica na opção “Close” e interrompe a execução do caso de uso.
Exceções	<ol style="list-style-type: none"> 1. O sistema deve emitir um erro em qualquer uma das seguintes opções e não prosseguir com a acção: <p>[FP_3] Não inserir a URL; [FP_4] Não seleccionar o projeto; [FP_5] Não seleccionar o plano de testes; [FP_6] Não seleccionar nenhum elemento da lista de MWBP</p>

A tabela 4.3 contém a descrição textual do caso de uso “Executar testes”. Este caso de uso consiste na execução dos testes de acessibilidade pretendidos e poderá ser efetuado pelos dois atores do sistema.

Testes automáticos de acessibilidade em aplicações móveis

Tabela 4.3: Descrição do caso de uso “Executar testes”

Nome	Executar Testes
Pré-condições	<ul style="list-style-type: none"> · Ter efetuado o Login no sistema · Ter um projeto de testes · Ter um Plano de testes · Ter um build
Objectivo	Este caso de teste descreve a atividade principal proposto: executar testes de acessibilidade de Webapps e depois disponibilizar os resultados no Testlink.
Fluxo principal	<ol style="list-style-type: none"> 1. Aceder a ferramenta de testes de acessibilidade ATAT 2. De seguida, o ator navega para a aba (TestPan) <ol style="list-style-type: none"> 2.1. Inserir a URL da Webapp 2.2. Procede-se a escolha do projeto de testes 2.3. Escolha do plano de testes 2.4. Selecionar os MWBP pretendidos 2.5. Submeter o formulário 3. Navegar para a aba testRunner 4. Escolher o test suite com o nome do MWBP escolhidos em 2.4 5. Clicar no botão “Run test” 6. Os resultados serão publicados na plataforma Testlink.
Fluxo Alternativo	1. [FP_1] O ator clica na opção “Close” e interrompe a execução do caso de uso.
Atores envolvidos	<ul style="list-style-type: none"> · Test manager · Tester
Exceções	O sistema deve emitir um erro em qualquer uma das seguintes opções:
	<ul style="list-style-type: none"> · [FP_1.2] O ator não insere uma URL; · [FP_1.2] URL não é válido: o sistema emite um erro; · [FP_1.3] O projeto não foi selecionado; · [FP_1.4] O plano de testes não foi selecionado; · [FP_1.5] Nenhuma norma foi selecionada;

A tabela 4.4 descreve o caso de uso para "criar projeto de testes"

Tabela 4.4: Criação de Projeto de testes

Nome	Criar projeto de testes
Pré-condições	Ter efetuado o Login no sistema
Objetivo	Criação de um projeto de testes de acessibilidade com menos passos do que a ferramenta Testlink, contemplando a criação automática dos respetivos elementos necessários ao projeto: test plan, build e test suite.
Fluxo principal	<ol style="list-style-type: none"> 1. Aceder a ferramenta de testes de acessibilidade ATAT 2. Navegar para a aba Test Project 3. Definir nome do projecto 4. Definir um prefixo do projeto 5. Escolher opções para criação automático de: <ol style="list-style-type: none"> 5.1) Criar do test plan; 5.2) Criar do build; 5.3) Criar do test Suite 7. Submeter a criação do projeto através do botão “Create Project”
Fuxo Alternativo	1. [FP_1] O ator clica na opção “Close” e interrompe a execução do caso de uso.
Atores envolvidos	Test manager Tester
Exceções	1. O sistema deve emitir um erro em qualquer uma das seguintes opções: [FP_3] Não inserção de nome do projeto; [FP_4] Não definição do prefixo; [FP_5] Não escolher nenhuma das opções do ponto 5;

De seguida apresenta-se a atividade inovadora nesta metodologia: a análise de robustez

4.4.1 Análise de Robustez

A segunda atividade da análise e desenho constitui a produção dos diagramas de robustez. ICONIX sugere o desenvolvimento desses diagramas antes, ou em paralelo, com a descrição textual dos casos de uso. A produção dos diagramas de robustez é uma das atividades mais importantes do processo de desenvolvimento ICONIX. De acordo com o ilustrado na figura 4.6 estes diagramas permitem ilustrar graficamente as interações entre os objetos que compõem um determinado caso de uso, deste modo, consegue-se diminuir possíveis erros na passagem da fase de análise (“o quê”) para a fase de desenho (“como”) [dSV10].

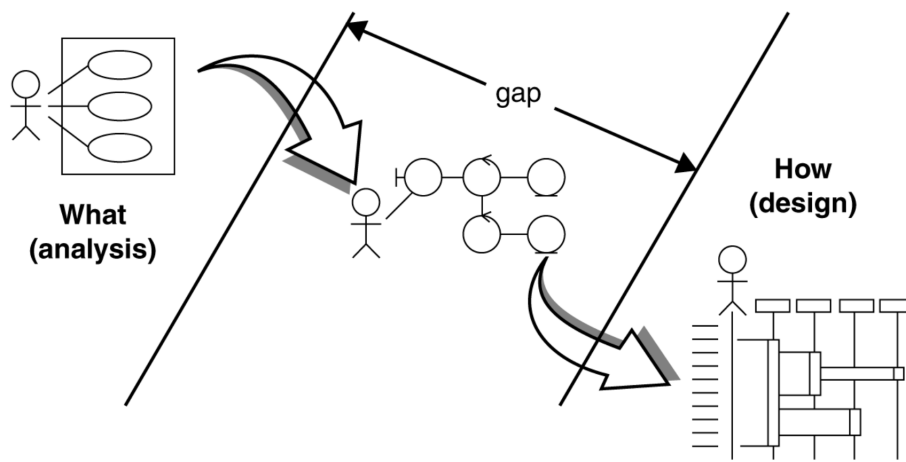


Figura 4.6: Importância dos diagramas de robustez

A análise de robustez utiliza três tipos de objetos (ver figura 4.7):) definidos no UML 1.3: objecto fronteira, objeto entidade e objeto controlo.

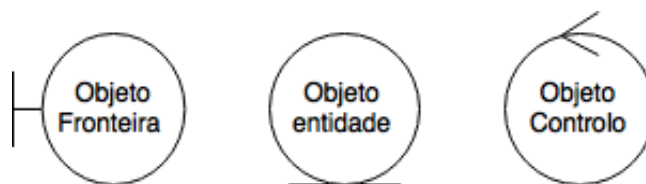


Figura 4.7: Os objetos da diagrama de robustez

- **Objetos fronteira** : permitem que os utilizadores interagem com o sistema;
- **Objetos entidade**: normalmente são objetos identificados pelo modelo de domínio;
- **Objetos controlo**: estes objetos funcionam como intermediários entre os objetos fronteira e os objetos entidade. Contem as regras de negócio, ou seja, pertencem ao modelo estático, o que potencia por um lado a independência das interfaces com os utilizadores e por outro, dos sistemas de base dados.

Na figura 4.8 pode-se consultar o diagrama de robustez do caso de uso “crias projeto de testes”. O diagrama é de alto nível, portanto não contém informação de em que parte da arquitetura do sistema ocorreram as operações definidas por exemplo pelos objetos controlo.

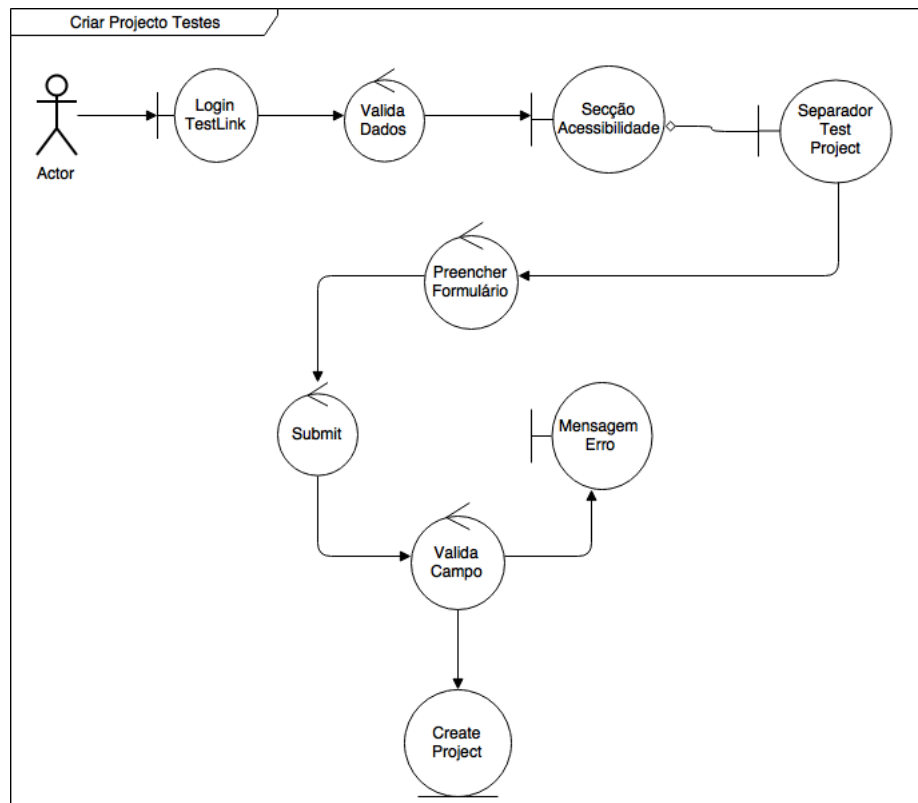


Figura 4.8: Diagrama de robustez para criação de projetos

O diagrama de robustez para a criação de plano de testes apresentada na figura 4.9, contém mais detalhes do que o anterior, pois é necessário disponibilizar informações dinamicamente.

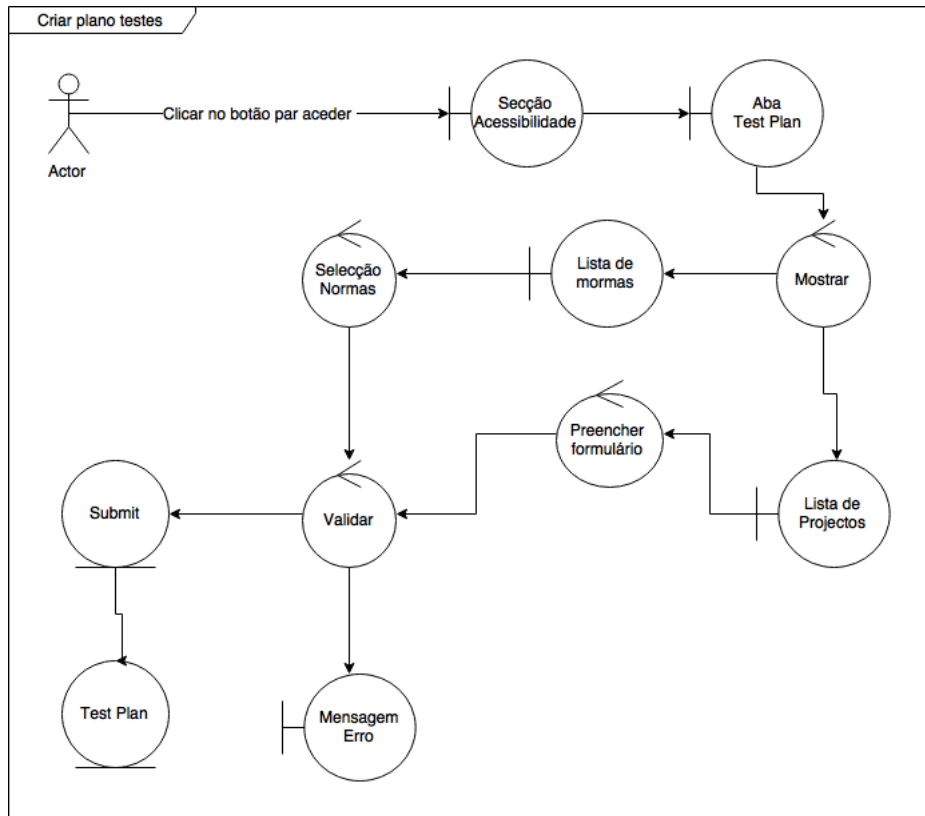


Figura 4.9: Diagrama de robustez para criação

Como podemos ver nas figuras 4.8 e 4.9 os diagramas de robustez produzidos apenas para os casos de uso já mencionados, revelaram a presença de pelo menos um novo objeto - Validar - que não estava descrito explicitamente nos casos de uso, mas que, poderá ser útil ao sistema. Nesse caso, procede-se à atualização do diagrama de classes com a criação de um novo método numa classe já existente para validar os dados ou adiciona-se uma nova classe.

Após a elaboração dos diagramas de robustez é necessário finalizar o modelo dinâmico, com a modelação do comportamento do sistema. Na próxima secção aborda-se a tarefa de desenho detalhado encarregue dessa modelação.

4.5 Desenho detalhado

Esta fase tem duas atividades principais: a definição detalhada do comportamento do sistema através de diagramas de sequência e o desenho da arquitetura do sistema e as respectivas tecnologias utilizadas.

4.5.1 Especificar o comportamento

A especificação do comportamento do sistema através de diagramas de sequência é a primeira atividade desta tarefa, consiste na utilização dos casos de uso com os respectivos diagramas de robustez e das descrições efetuadas na tarefa de análise e desenho preliminar, para elaboração dos mesmos. ICONIX sugere que a preocupação não deve ser a representação detalhada e completa do fluxo de mensagens, mas sim uma visão de alto nível sobre o comportamento[DSV10]. ICONIX recomenda que o comportamento de um caso de uso ilustrado pelo seu diagrama de

robustez, deve ser, nesta atividade detalhado através de um diagrama de sequência, onde se utiliza os objetos e atores definidos neste diagrama para demonstrar o fluxo de mensagens trocados entre si [dSV10]. A figura 4.10 exemplifica essa recomendação.

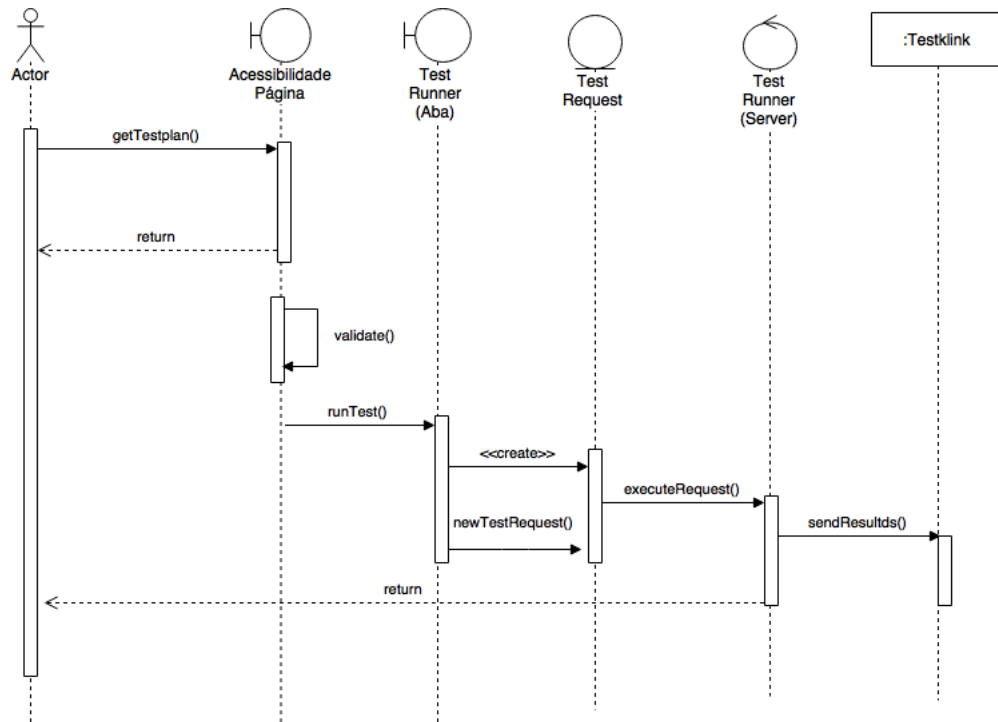


Figura 4.10: Diagrama de sequência do caso de uso “Executar teste”.

De notar que este diagrama (Figura 4.10) não apresenta detalhes sobre a ligação entre o cliente e o servidor, e a forma como o servidor de testes trata a requisição de testes, a sua execução e envio dos resultados para o *Testlink*. Convém referir que a aplicação cliente conterá uma aba “Test Runner”, que representa o objeto fronteira para fazer requisições de testes via web-services. Assim que o serviço é invocado é instanciado um objecto *TestRequest* que servirá de parâmetro para o objeto controlo *Test Runner* alocado no servidor, responsável pela execução instanciação das classes e da chamada dos métodos necessários para desencadear todo o processo de testes. O diagrama de sequência do caso de uso “criar plano de teste” elaborado do modo tradicional da UML, portanto não se baseia no diagrama de robustez introduzida pelo ICONIX, é apresentada na figura 4.11.

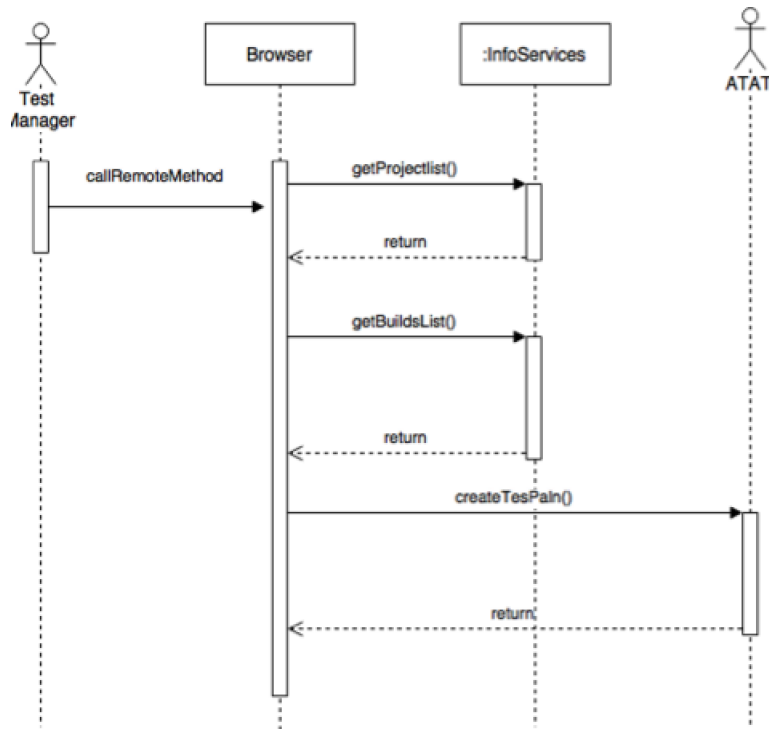


Figura 4.11: Diagrama de sequência da criação de um plano de teste.

Este diagrama não manifesta toda a infraestrutura por detrás do processo, e apresenta em alto nível o fluxo de mensagens para a criação de plano de testes de acessibilidade.

A atividade a seguir seria a atualização do modelo de domínio, porém, omite-se essa parte. No entanto, será apresentado, parte do diagrama de classes que nesta metodologia é a evolução do modelo de domínio.

Na próxima secção apresenta-se a arquitetura geral da ferramenta ATAT.

4.5.2 Arquitetura geral da ATAT

O processo de automação de testes requer um enorme esforço na área de engenharia de software, tal como os projetos de software tradicionais. Neste sentido, propõe-se uma arquitetura distribuída em quatro camadas para auxiliar na criação de um nível de abstração, que por sua vez, aumenta significativamente a flexibilidade, escalabilidade e qualidade do serviço que a ferramenta de testes automáticos fornece. Na figura 4.12 pode-se ter uma visão geral do sistema e da arquitetura cliente/servidor proposta. Para isso, definiu-se as seguintes camadas: camada de dados, camada aplicação, camada Web e a camada de apresentação.

Testes automáticos de acessibilidade em aplicações móveis

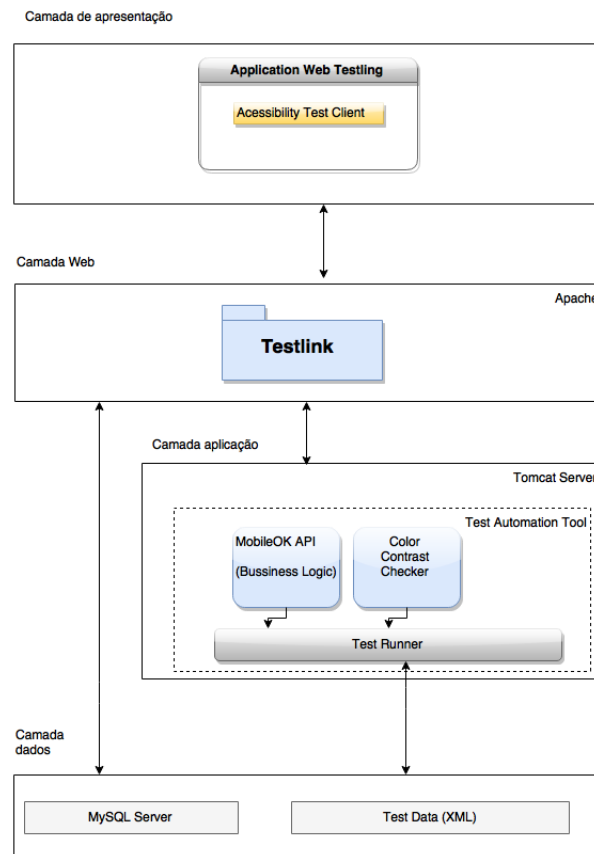


Figura 4.12: Figura XX: Arquitetura geral do sistema(a).

A camada de apresentação é caracterizada pelo cliente Web que interage com as camadas inferiores. Esta é a camada que providencia Graphical User Interface (GUI) deve-se focar exclusivamente no tratamento e apresentação da informação, garantindo um user experience simples e eficiente [CMV⁺13] [Day14]. É essencial garantir que a GUI seja desenvolvida de acordo com normas de usabilidade e acessibilidade. Na camada Web situa-se a plataforma Testlink que

controla o armazenamento dos projetos de testes de acessibilidade e que é responsável pela geração da GUI para camada superior. Os serviços da camada Web são fornecidos pelo servidor Apache. A camada de aplicação representa o núcleo operacional do sistema. É onde se faz todo processamento, disponibilização de recursos e serviços. Através do servidor Tomcat, esta camada implementa todos os mecanismos necessários para receber as requisições de testes, executa-las e disponibilizar os resultados na camada imediatamente superior. Uma descrição mais detalhada desta camada é apresentada no capítulo 5.

A camada de dados é responsável pela persistência dos dados. A execução dos testes pode gerar uma quantidade significativa de dados que precisam de ser armazenados. No nosso caso, usamos o formato padronizado pela W3C: EXtensible Markup Language (XML) para estruturar e armazenar os resultados dos testes. Nesta camada também está o MySQL Server, que é o servidor de base de dados utilizada pela plataforma Testlink. Um visão mais formal da arquitetura é apresentada na figura 4.13 através de um diagrama de componentes.

Testes automáticos de acessibilidade em aplicações móveis

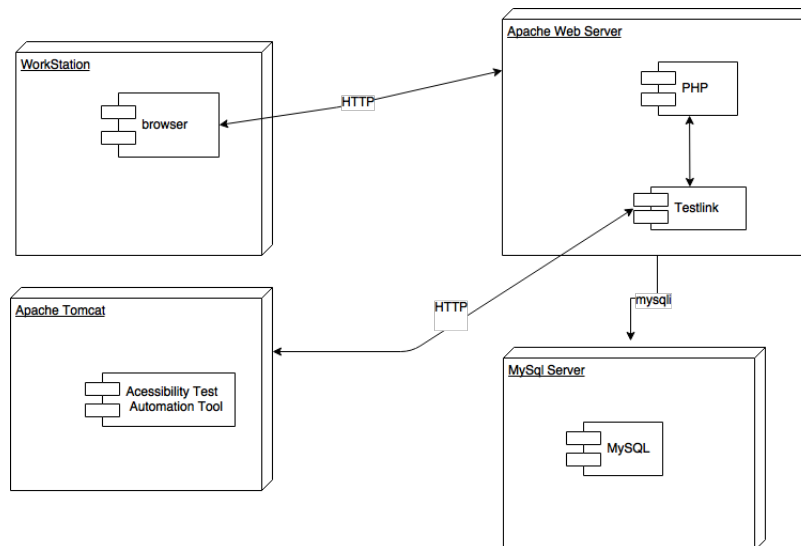


Figura 4.13: Arquitetura geral do sistema (b).

A atividade seguinte constitui a elaboração do diagrama de classes que servirá de suporte à implementação do sistema na tarefa seguinte.

4.5.3 Diagrama de Classes

Após a conclusão das tarefas acima descritas, procede-se à elaboração do diagrama de classes (ver figura 4.14).

Testes automáticos de acessibilidade em aplicações móveis

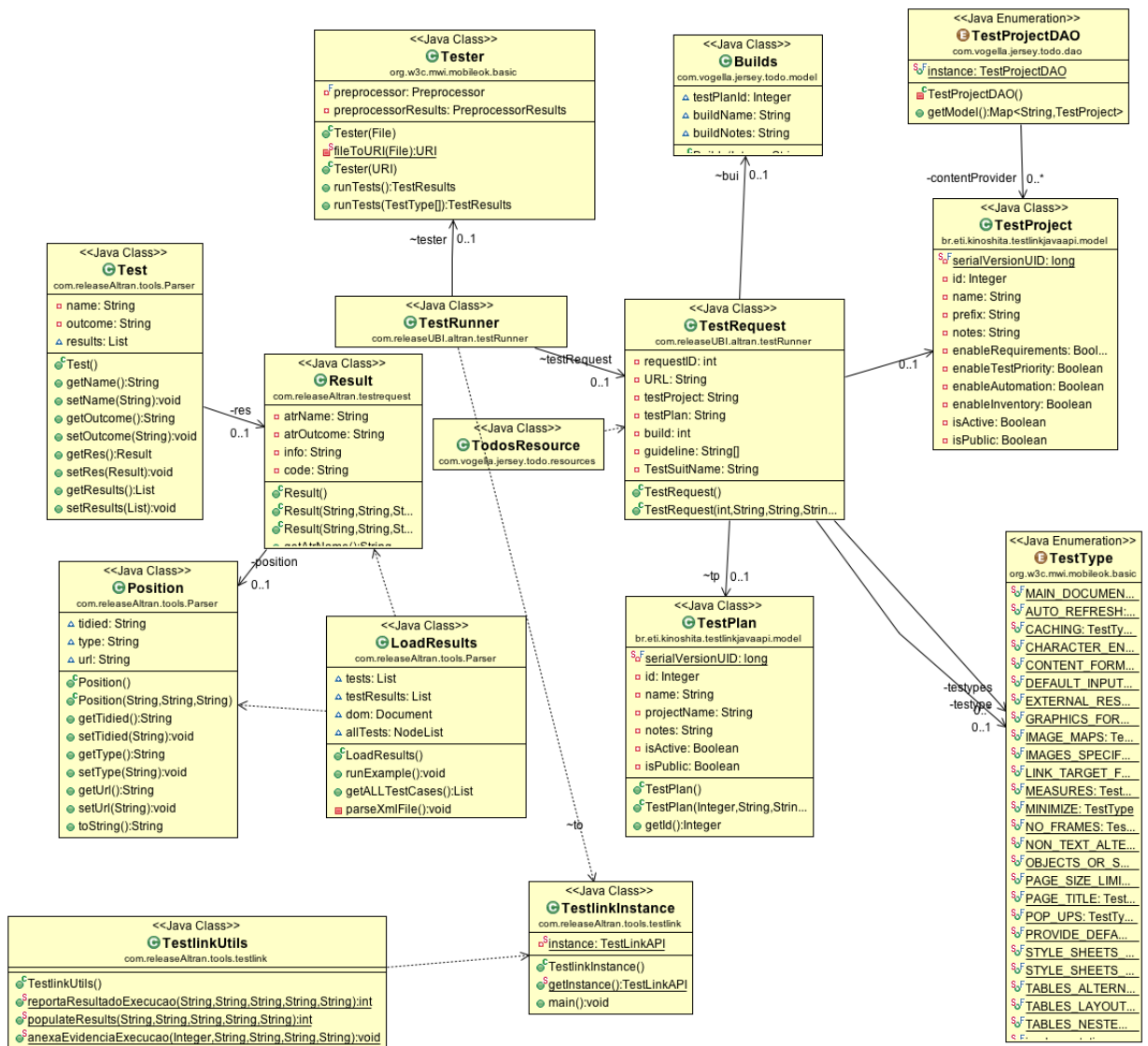


Figura 4.14: Diagrama de classes parcial.

Devido à complexidade e grande dimensão do diagrama de todas as classes num todo apresenta-se apenas uma parte do mesmo. Neste contexto, a apresentação da sua totalidade e com todos os detalhes da arquitetura acima descrito, resultaria numa figura impercetível. No entanto, classes importantes para o sistema, tal como a classe *TestRequest*, figuram na imagem apresentada como forma de exemplificar a sua importância. Como se constata através das suas relações com classes como *TestProject*, *TestPlan*, *Tester* e *TestRunner*. Outra relação importante de realçar, é entre as classes *TestRunner* e a *TestlinkInstance*. A classe *TestlinkInstance* disponibiliza à classe *TestRunner*, os métodos necessários para o envio dos dados para a plataforma de gestão de testes *Testlink*.

A fase implementação e testes representam o culminar de todo o processo de desenvolvimento. Considera-se que pela importância dessa fase, a sua descrição seria elaborada num capítulo separado, neste caso, no capítulo 5.

4.6 Conclusão

Pretendeu-se neste capítulo fazer uma análise, desenho e arquitetura da ferramenta, seguindo uma metodologia de desenvolvimento de software iterativo e incremental, ainda neste capítulo, abordou-se a importância da análise de requisitos tem na definição objetiva do produto. Fez-se uma descrição da metodologia ICONIX e sua aplicação no processo de desenvolvimento da ferramenta ATAT.

As análises descritas resultantes deste processo, bem como, os artefactos produzidos, servirão como documentação para a próxima tarefa definida pelo ICONIX e que será descrita no próximo capítulo.

Capítulo 5

Implementação da ferramenta ATAT

5.1 Introdução

No capítulo anterior fez-se a análise do sistema a ser implementado. Neste capítulo descreve-se a tarefa de implementação sugerida pela metodologia ICONIX, bem como as ferramentas e tecnologias utilizadas.

Inicialmente aborda-se a implementação da aplicação cliente e do servidor de testes, na qual se descreve as principais ferramentas utilizadas. De seguida é demonstrado o processo de criação e execução de testes automáticos de acessibilidade. Por último, apresentam-se alguns resultados da prova de conceito do sistema, aplicada à aplicação da Axa Banque.

5.2 Implementação

A solução proposta para o problema apresentado anteriormente é dividida em duas partes: o cliente e o servidor. Nesta secção aborda-se as suas implementações e o módulo de comunicação entre ambos.

5.2.1 Implementação do cliente

A aplicação cliente desenvolvida na camada de apresentação da arquitetura descrita no capítulo anterior e incorporada no *Testlink* como módulo foi desenvolvida seguindo um padrão arquitetural *Model View Controller* (MVC). Apesar do *Testlink* não seguir nenhum padrão, considerou-se que seria benéfico obter a separação do acesso a dados (*Model*), interface gráfica (*View*) e lógica de negócio (*Controller*), a fim de, aproveitar as vantagens da sua independência.

5.2.2 Implementação do servidor

O Servidor foi desenvolvido seguindo o paradigma orientado a objeto com a tecnologia *Java Enterprise Edition (javaEE) 7*, sendo que o servidor consiste num projeto *JavaEE Dynamic Web page*. A figura 5.1 mostra a interligação dos principais pacotes que suportam o desenvolvimento do protótipo.

Através de um diagrama de componentes, apresentamos nesta secção a arquitetura geral do servidor, na qual mostramos as dependências do pacote *TestRunner*.

Testes automáticos de acessibilidade em aplicações móveis

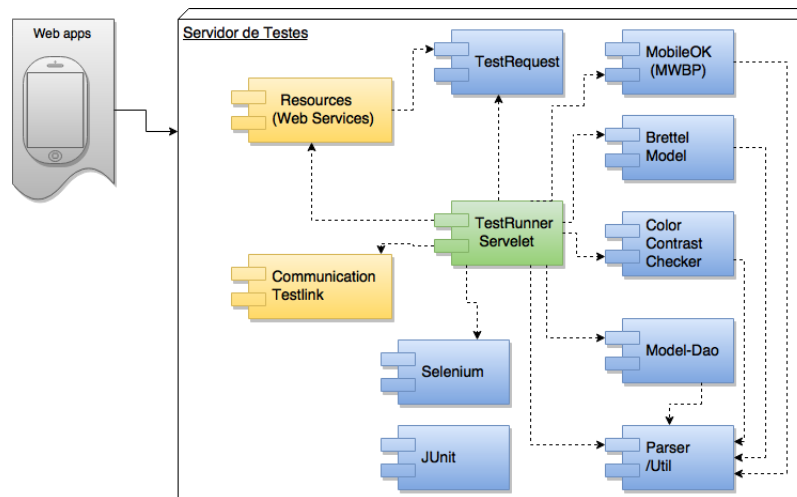


Figura 5.1: Arquitetura do servidor de testes.

De acordo com a figura 5.1 foram implementadas os seguintes pacotes:

- **TestRequest** - Este pacote contém a classe com o mesmo nome (TestRequest) responsável pelo encapsulamento num único objecto Plain Old Java Object (POJO) e transformação dos dados da requisição dos testes, em tipos de dados aceites pelo pacote TestRunner;
- *Communication Testlink* - como o próprio nome indica é o pacote onde se implementa as classes e métodos, com base na Application Programming Interface (API) Testlink java, para a comunicação com a plataforma Testlink desenvolvida com tecnologia Hypertext Preprocessor (PHP).
- *Resources (Web Services)*- neste pacote encontram-se os recursos disponíveis no servidor. Este pacote consta na definição da configuração do servlet.
- *Color Contrast Checker* - é o pacote onde é implementada uma ferramenta baseada em algoritmos definidos pela W3C, para a verificação através da luminosidade e contraste de cores do *background* e do *foreground* de modo a permitirem a legibilidade de textos.
- *Model-DAO* - este pacote é responsável pelo fornecimento de todos os modelos das classes. Segue o padrão Data Access Object (DAO) que privilegia a persistência dos dados
- *Parser/Util* - este pacote contém o *parser* desenvolvido para fazer o *parsing* dos dados em XML para a sua representação em DAO.
- *JUnit* - este pacote contém os testes unitários das principais classes que compõem o sistema.
- *Selenium* - é responsável pela interação inicial do servidor com o ambiente de testes (nesse caso um navegador Web). Através desse pacote automatiza-se o login para que se proceda ao teste de toda a Webapp.
- *TestRunner* - este pacote é responsável pela execução efetiva dos testes. A figura 5.1 ilustra as suas dependências. Toda o processamento é coordenado por este pacote, que pela instanciação de classes dos outros pacotes fornece o serviço de testes.

O funcionamento do servidor obedece à seguinte descrição: receber uma requisição através do *Web service* disponibilizado no pacote *Resources*, cria-se uma instância da classe *TestRequest*

com os dados provenientes do cliente. De seguida, este objeto é utilizado pelo pacote *TestRunner* para a execução dos testes e o envio dos resultados para o *Testlink*. Para o efeito, foi desenvolvido um módulo de comunicação bidirecional.

Na figura 5.2 ilustra-se a comunicação entre a plataforma *TestLink* (PHP) e o servidor de testes (Java).

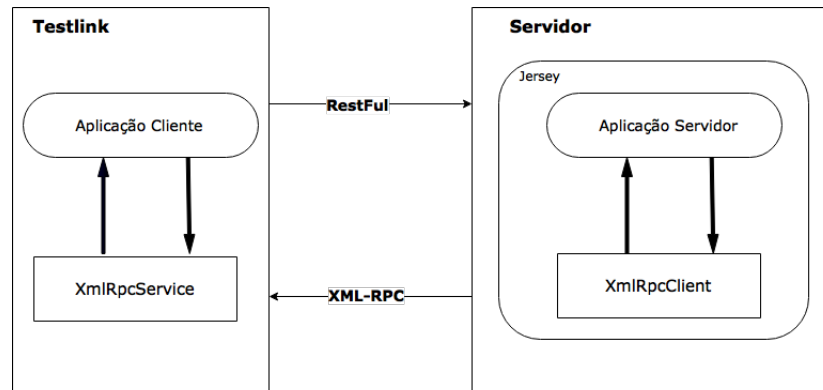


Figura 5.2: Comunicação bidirecional

A troca de mensagens entre o *Testlink* e o servidor de testes implica a implementação de mecanismos para permitir a comunicação bidirecional. No sentido cliente - servidor (*Testlink* - Servidor de testes) é realizada através de Web Services disponibilizados pelo servidor e XML-Remote Procedure Call (RPC). Em geral, o cliente faz um pedido *RESTful* ao servidor, que por sua vez pode responder de duas formas:

1. Por XML- RPC na qual os resultados dos testes são integrados nos projetos já existentes;
2. Os dados são disponibilizados em XML e/ou JSON, o que facilita a escalabilidade e portabilidade. Desde modo, os resultados ficam disponíveis para qualquer plataforma tecnológica pretendida.

No sentido servidor-cliente a comunicação é feita através do servidor XML-RPC o que permite a execução de métodos do sistema via uma chamada de um procedimento remoto. Os Web Service (WS) constituem a forma mais comum de comunicação entre computadores, analisou-se as arquiteturas de WS que melhor se adequam ao protótipo. Foram analisados o Simple Object Access Protocol (SOAP) e o REST. Pela simplicidade, optou-se pela *REST* que suporta a utilização na sua arquitetura dos mesmos métodos do protocolo Hypertext Transfer Protocol (HTTP) (*post*, *get*, *put* e *delete*). Nesta arquitetura tudo é tratado como um recurso, acedido via *Uniform Resource Identifiers* (URI), tipicamente descritos como ligações Web. Um recurso pode ser visto como um objecto no paradigma orientado a objetos, ou como uma entidade numa base de dados.

A disponibilização de serviços via HTTP requer a utilização de um *Web container*. Neste sentido analisou-se o *GlassFish*, o *JBoss*, e o *Tomcat*. Optou-se pela última opção, apesar de *Glassfish* e *Jboss* disponibilizarem mais funcionalidades que o *Tomcat*, considerou-se que para este caso em concreto, este servidor de aplicações seria o mais indicado.

5.2.3 Principais ferramentas utilizadas

Jersey é a implementação de referência, suportada pela API JAX-RS 2.0 já incorporada no Java 7, que suporta as anotações definidas no JSR 311, tornando mais fácil o desenvolvimento de *Web services RESTful* com a linguagem de programação Java. Os recursos são identificados por URIs, que proporcionam um espaço de endereçamento global de recursos e de descoberta de serviço.

A implementação Jersey fornece uma biblioteca para implementar *Web Services* num *Java Web Container* através de um *servlet*. Este *servlet* analisa a solicitação via HTTP e seleciona a classe correta e o correspondente método para responder a este pedido. Esta seleção é baseada em anotações na classe e métodos.

De referir que o Jersey fornece a possibilidade de implementação de *Servlets* que fazem a procura das classes predefinidas, e assim identificar recursos *RESTful*. Na configuração do *servlet* no ficheiro *web.xml* são definidos os pacotes onde o Jersey deve procurar os recursos, sendo estas, classes ou métodos.

Paralelamente ao desenvolvimento do sistema utilizou-se *JUnit* para a realização dos testes unitários, de forma a garantir que os componentes importantes do sistema funcionassem de acordo com o esperado. *JUnit* é um *framework open-source*, criado por Erich Gamma e Kent Beck, que facilita à criação de testes unitários automatizados na linguagem de programação Java.

De seguida apresenta-se um exemplo de funcionamento do sistema.

5.3 Criação e execução de testes automáticos

Nesta secção é apresentado o protótipo da aplicação desenvolvida e a sua utilização nos testes de acessibilidade num produto da Axa Banque. O ciclo de execução do protótipo proposto segue as seguintes etapas:

1. Preparação dos testes;
2. Criação de um plano de testes de acessibilidade;
3. Execução dos testes.

5.3.1 Preparação dos testes

Após uma análise da plataforma *Testlink* chegou-se à conclusão que esta, contém alguns problemas de usabilidade e acessibilidade que dificultam a perceção e navegabilidade (mencionadas no capítulo 3), o que prejudica a criação das pré-condições para execução dos testes, descritas no capítulo 4. Na figura 5.3 encontra-se a página inicial do *Testlink*, que demonstra a necessidade de execução de quatro passos ilustrados numericamente pela ordem de precedências.

Testes automáticos de acessibilidade em aplicações móveis

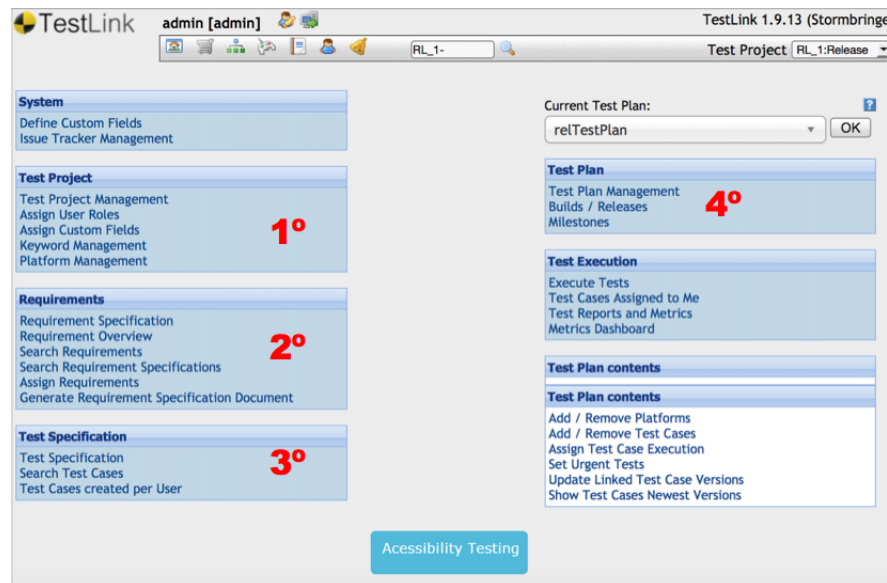


Figura 5.3: Página inicial do Testlink

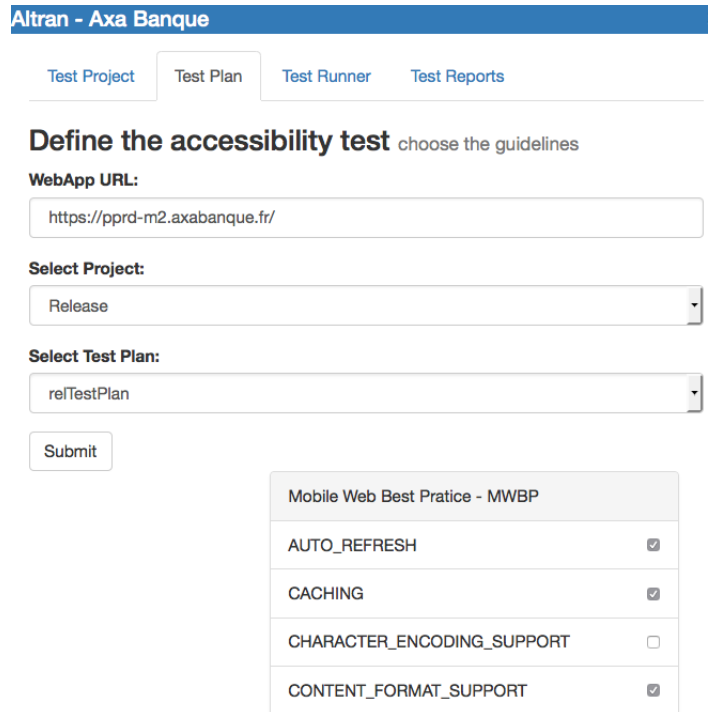
Para ultrapassar este problema e agilizar o processo, criaram-se as funcionalidades necessárias a fim de que, numa única interface (ver figura 5.3), se proceda à criação das pré-condições mencionadas anteriormente. Esta interface é acedida através do módulo adicionado ao *Testlink*, tal como, especificado pelos objetivos descritos no capítulo 1.

Figura 5.4: Criação de um projeto de testes completo

A figura 5.4 demonstra a interface única, para a criação de um projeto de testes e as suas dependências. Ao habilitar as *checkboxs*, o sistema criará automaticamente um plano de teste, um *build* e *test suite* com dados predefinidos. Posteriormente à criação do projeto, exemplifica-se na próxima etapa a planificação dos testes de acessibilidade.

5.3.2 Criação de um plano de testes de acessibilidade

Segundo o proposto, a figura 5.5 (imagem parcial) exemplifica a criação de um plano de testes de acessibilidade. Conforme se pode verificar deve-se inserir a URL da *WebApp* que irá ser testada, selecionar o projeto, o respetivo plano de teste e os MWBP.



Altran - Axa Banque

Test Project | **Test Plan** | Test Runner | Test Reports

Define the accessibility test choose the guidelines

WebApp URL:

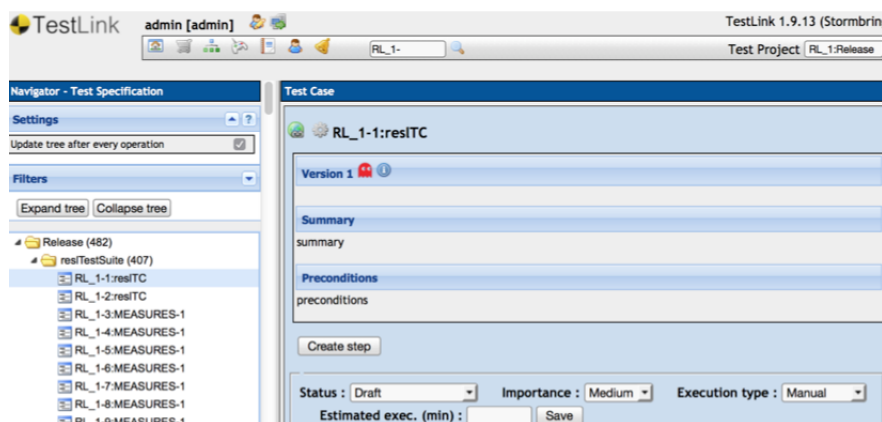
Select Project:

Select Test Plan:

Mobile Web Best Pratices - MWBP	
AUTO_REFRESH	<input checked="" type="checkbox"/>
CACHING	<input checked="" type="checkbox"/>
CHARACTER_ENCODING_SUPPORT	<input type="checkbox"/>
CONTENT_FORMAT_SUPPORT	<input checked="" type="checkbox"/>

Figura 5.5: Criação de um projeto de testes completo

Após preenchimento e submissão do formulário, é criado, no plano de testes "resTestPlan", um *test suite* com os testes automatizados selecionados anteriormente, sendo que este pode conter ou não testes manuais. O test suite criado anteriormente está representado na figura 5.6 como se pode verificar a opção *Execution Type* está definida como manual, que certifica a existência de casos de testes manuais, neste caso, dois: RL_1-1:resTC e o RL_1-2:resTC, que antecedem os testes automatizados. O *test suite* relTestSuite contem no total 407 casos de testes, sendo que dois são manuais e os restantes gerados automaticamente pela ferramenta de automatização de testes.



TestLink admin [admin] TestLink 1.9.13 (Stormbrink)

Test Project: RL_1:Release

Navigator - Test Specification

Settings: Update tree after every operation ☒

Filters: Expand tree | Collapse tree

- Release (482)
 - relTestSuite (407)
 - RL_1-1:resTC
 - RL_1-2:resTC
 - RL_1-3:MEASURES-1
 - RL_1-4:MEASURES-1
 - RL_1-5:MEASURES-1
 - RL_1-6:MEASURES-1
 - RL_1-7:MEASURES-1
 - RL_1-8:MEASURES-1
 - RL_1-9:MEASURES-1

Test Case

RL_1-1:resTC

Version 1

Summary

Preconditions

Create step

Status: Draft Importance: Medium Execution type: Manual

Estimated exec. (min): Save

Figura 5.6: Criação de teste suite no Testlink

Testes automáticos de acessibilidade em aplicações móveis

Por outro lado, se não for indicado um *test Suite*, este será criado automaticamente com o nome do MWBP escolhido. Um exemplo está ilustrado na figura 5.7 onde se certifica a existência de alguns *tests suite*, nomeadamente o “Measures”, “Auto_refresh” e “Caching”, com o nome e os respetivos casos de teste gerados automaticamente.

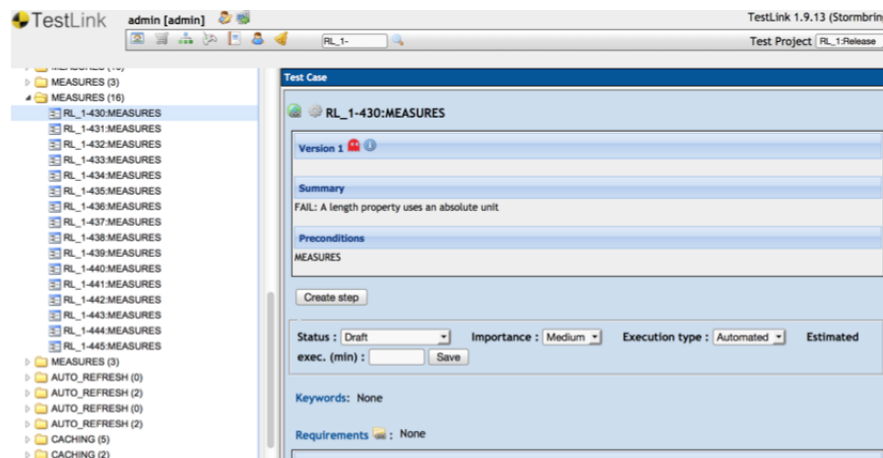


Figura 5.7: Criação de teste suite no Testlink

Após a fase de preparação e criação de um plano de testes de acessibilidade, ocorre a sua execução, que se descreve de seguida.

5.3.3 Execução dos testes

A execução automática de testes automatizados constitui o objetivo principal do sistema. Esse objetivo poderá ser alcançado na interface ilustrada na figura 5.8, em que o TestID, o TestPlan, Project e MWBP foram escolhidos na etapa anterior.

Test ID	Test Plan	Project	Test Runner
MEASURES	1691	1690	<button>Run Test</button>
AUTO_REFRESH	1691	1690	<button>Run Test</button>
CACHING	1691	1690	<button>Run Test</button>
AUTO_REFRESH	1691	1690	<button>Run Test</button>
PROVIDE_DEFAULTS	2	1	<button>Run Test</button>

Figura 5.8: Interface Test Runner

O botão “Run Test” está associado às Web services disponibilizadas pelo servidor, portanto, é responsável pela requisição de todo e qualquer teste. Conforme referido no capítulo 4 essa requisição é feita via RESTful. Nesta fase, o processamento passa a ser realizado no servidor, que após a sua conclusão adiciona ao projeto identificado pelo seu ID, os resultados da execução. Esta operação ocorre em baixo nível.

O exemplo descrito na figura acima resulta da execução “MEASURES” (o 1º testID). Esta MWBP testa a WebApp em relação ao tipo de medidas que utiliza na definição dos seus elementos em HTML5 ou CSS. Por outras palavras, testa-se se as unidades de medida, tais como pixel(px), centímetro (cm) e polegada(in) são absolutas ou relativas, tal como, a percentagem (%). Com

medidas relativas, a Webapp adapta-se facilmente aos diferentes tamanhos dos ecrãs dos dispositivos móveis. Conforme foi descrito no capítulo 2, o tamanho dos ecrãs constitui um desafio aos testes de usabilidade e acessibilidade.

Na próxima secção serão discutidos os resultados obtidos dos testes de acessibilidade à aplicação em estudo.

5.4 Resultados dos testes acessibilidade do caso de estudo

Apresenta-se nesta secção os resultados da execução dos testes à aplicação da Axa Banque. Utilizou-se o ambiente de testes funcionais que é disponibilizado via URL à equipa pela Axa Banque e um cliente fictício, neste caso, o M.Stephane Dumond.

A aplicação consiste em uma Web app (referida no capítulo 2) disponibilizada via URL pela Axa Banque, para efeitos de testes funcionais(ver figura 5.9). Utilizou-se o mesmo ambiente para os testes de acessibilidade.



Figura 5.9: Aplicação da Axa Banque

Para isso foi necessário, numa fase inicial, a utilização de testes funcionais automatizadas, para fazer o *login* a aplicação, e posteriormente ter acesso os outros conteúdos dentro da aplicação. Utilizou-se a ferramenta *Selenium* conjuntamente com o *framework* de testes unitários JUnit, para o efeito. Na figura 5.11 exemplifica o *login* na aplicação, na qual mostra-se, através de um *assert* ao nome do cliente acima referido.

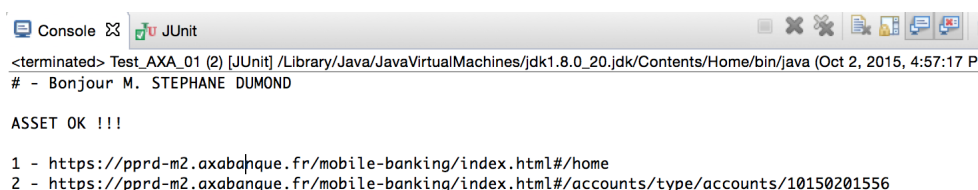


Figura 5.10: Login à aplicação da Axa Banque

Testes automáticos de acessibilidade em aplicações móveis

Constata-se pela figura 5.11, que representa uma execução do ATAT que o nome se procedeu ao *login* a aplicação com sucesso, e posteriormente, inicia-se a operação de recolha de informação da Webapp. Neste caso das URLs, para testar individualmente.

No canto superior da figura 5.11 verifica-se que o nome do M.Stephane Dumond coincide com o resultado da ATAT.

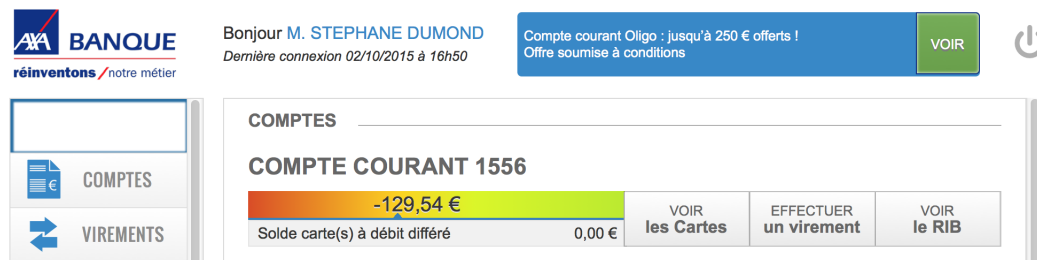


Figura 5.11: Logins

Os resultados dos testes são definidos em três categorias de acordo com a API MoibleOK: PASS, FAIL e WARN.

- PASS: acontece quando todos os casos de testes gerados automaticamente estão de acordo com as *guidelines* que suportam MWBP;
- FAIL: ocorre quando pelo menos um dos casos de testes falha;
- WARN: é apenas uma advertência de situações que não constituem FAIL mas podia ser melhoradas.

Os resultados globais que resultam da execução de todo o processo acima descrito são apresentados na tabela 5.1.

Tabela 5.1: Resultados gerais dos testes de acessibilidade do caso de estudo

Mobile Web Best Practice	Outcome
Auto_Refresh	PASS
Caching	FAIL
Character_Encoding_Support	FAIL
Content_Format_Support	FAIL
Default_Input_Mode	PASS
External_Resources	FAIL
Graphics_For_Spacing	FAIL
Image_Maps	PASS
Images_Specify_Size	PASS
Link_Target_Format	PASS
Measures	FAIL
Minimize	PASS
No_Frames	PASS
Non_Text_Alternatives	PASS
Objects_Or_Script	FAIL
Page_Size_Limit	FAIL
Page_Title	PASS
Pop_Ups	PASS
Provide_Defaults	PASS
Style_Sheets_Support	PASS
Style_Sheets_Use	FAIL
Tables_Alternatives	PASS
Tables_Layout	PASS
Tables_Nested	PASS

Num total de vinte e quatro MWBP testados, quinze tiveram o resultado final como *PASS* e as restantes nove foram *FAIL*. Os testes de acessibilidade são subjetivos e em muitos casos difíceis de executar manualmente. Por exemplo, em casos em que se deve testar se a Web app utiliza o recurso de *Auto_Refresh*, torna-se um processo moroso percorrer as linhas de código manualmente e fazer tal verificação. No entanto, essa tarefa é mais facilmente executada de forma automatizada o que prova a necessidade de ferramentas automáticas para esse tipo de testes. Para além dessa necessidade, é importante garantir, e possivelmente, quantificar a sua eficácia. De seguida apresentamos uma forma de testar por meio dos testes manuais, se o prototipo desenvolvido funciona de acordo com o esperado.

5.4.1 Testes manuais para provar o funcionamento

Para comprovar a eficácia do sistema foram executados manualmente alguns dos testes aos MWBP executados automaticamente. Este resultados são apresentados na tabela 5.2

Tabela 5.2: Resultados da execução manual

Testes Manuais	Resultado Esperado	Resultado Obtido
No_Frame	Pass	Pass
Tables_Alternatives	Pass	Pass
Tables_Layout	Pass	Pass
Tables_nested	Pass	Pass

Estes resultados consistiram em evidenciar manualmente, a existência no código fonte da página inicial da Web app em estudo, de elementos que expõe a aplicação à falhas de acessibili-

Testes automáticos de acessibilidade em aplicações móveis

dade.

Constata-se através da tabela 5.1 e da tabela 5.2 que os resultados obtidos automaticamente e os obtidos manualmente coincidem. Desde modo, e para os testes manuais executados, podemos afirmar que o nosso prototipo apresentou uma eficácia na ordem dos 100%. De referir que para se certificar e quantificar todo o sistema desenvolvido seria necessário um trabalho mais exaustivo.

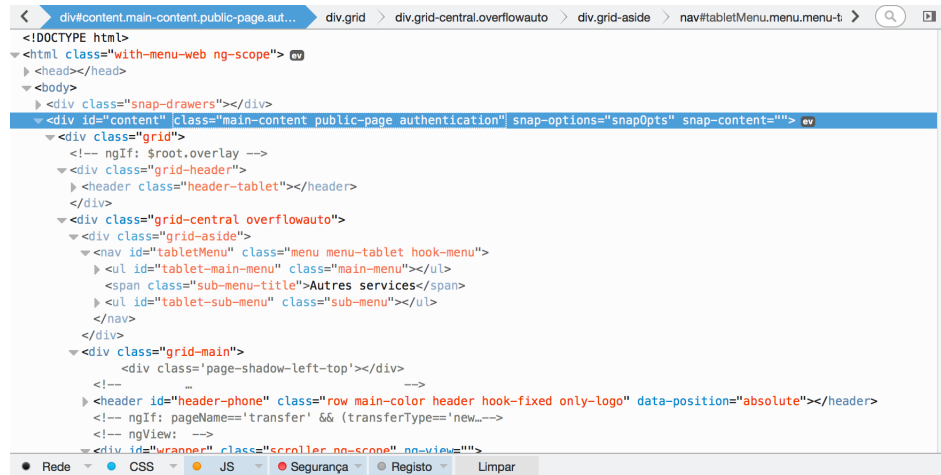


Figura 5.12: Exemplo de execução testes manualmente

A figura 5.12 ilustra um exemplo dos testes manuais efetuados, onde se pode confirmar que a estrutura principal da página, não foi definida em tabelas e nem com a utilização de *frames*, portanto os testes devem ser definidos como *PASS*. Estes elementos são tidos como obsoletos e a sua utilização é desaconselhada pela W3C [W3C14]. Ao invés, utilizou-se elementos da *HTML5*, *div* e *nav* para a definição da estrutura da página, o que confere a página uma acessibilidade e capacidade de adaptabilidade às diferentes tamanhos e resoluções de ecrãs.

5.4.2 Resultados detalhados dos testes de acessibilidade

Na tabela 5.3 apresenta-se os resultados de forma mais detalhada, onde se pode ver o tempo de execução de cada MWBP, os resultados e as quantidades de casos de testes gerados.

Tabela 5.3: Resultados detalhados dos testes

MWBP	Tempo de Execução	Resultado	Qtd. Casos de testes
NO_FRAMES	35,803	PASS	2
		Warn	0
		FAIL	0
AUTO_REFRESH	39,129	PASS	2
		War	0
		FAIL	0
CHARACTER_ENCODING_SUPPORT	33,421	PASS	0
		Warn	0
		FAIL	1
CONTENT_FORMAT_SUPPORT : FAIL	39,825	PASS	0
		Warn	1

Testes automáticos de acessibilidade em aplicações móveis

		FAIL	15
DEFAULT_INPUT_MODE	37,6	PASS	0
		Warn	0
		FAIL	1
EXTERNAL_RESOURCES	35,652	PASS	0
		Warn	1
		FAIL	11
GRAPHICS_FOR_SPACING	36,632	PASS	0
		Warn	0
		FAIL	12
IMAGE_MAPS	33,261	PASS	0
		Warn	0
		FAIL	1
IMAGES_SPECIFY_SIZE	33,195	PASS	2
		Warn	0
		FAIL	0
LINK_TARGET_FORMAT	49,996	PASS	2
		Warn	0
		FAIL	0
MAIN_DOCUMENT	44,497	PASS	0
		Warn	0
		FAIL	1
MEASURES	48,456	PASS	0
		Warn	0
		FAIL	404
MINIMIZE	46,888	PASS	2
		Warn	0
		FAIL	0
NO_FRAMES	35,42	PASS	2
		Warn	0
		FAIL	0
NON_TEXT_ALTERNATIVES	47,44	PASS	13
		Warn	0
		FAIL	0
OBJECTS_OR_SCRIPT	33,369	PASS	0
		Warn	5
		FAIL	2
PAGE_SIZE_LIMIT	33,180	PASS	0
		Warn	0
		FAIL	4
PAGE_TITLE	32,965	PASS	0
		Warn	0
		FAIL	0
POP_UPS	31,961	PASS	0
		Warn	0

		FAIL	2
PROVIDE_DEFAULTS	34,749	PASS	2
		Warn	0
		FAIL	0
STYLE_SHEETS_SUPPORT	45,328	PASS	0
		Warn	681
		FAIL	1
TABLES_ALTERNATIVES	30,573	PASS	2
		Warn	0
		FAIL	0
TABLES_LAYOUT	31,606	PASS	2
		Warn	0
		FAIL	0
TABLES_NESTED	36,667	PASS	2
		Warn	0
		FAIL	0

De uma forma mais sucinta, na tabela ?? demonstra os tempos de execução. Este parâmetro constitui um elemento importante nos testes. Em casos em que os testes demoram muito tempo, desencorajam a sua execução no processo de desenvolvimento.

Tabela 5.4: Tempo de execução dos testes

	Tempo total de execução	Qtd. Total de casos de testes
Segundos	907.613	729
Minutos	15.13	
Médio	37.9	

Na tabela acima referida constata-se que o tempo total de execução é de 907.613 segundos, o que equivale a um total de 15.13 minutos. Apesar de 15 minutos para uma equipa de testes poderá não significar muito tempo, para quem desenvolve o produto é um tempo exorbitante[Mar]. O uso de técnicas de paralelização de processos poderia ser utilizada para diminuir o tempo global da execução dos testes.

5.5 Conclusão

Neste capítulo está concentrado o maior esforço do trabalho. A implementação foi realizada de acordo com os artefactos desenvolvidos o capítulo anterior. Escolheu-se a linguagem de programação Java para implementação da camada de negócio da aplicação e a linguagem Web PHP, para desenvolver a aplicação da camada de Web.

No próximo capítulo abordamos as conclusões e os trabalhos futuros.

Capítulo 6

Conclusão

A problemática da acessibilidade digital é transversal a toda a sociedade, o que a torna num elemento que permite garantir a universalidade dos produtos digitais. A popularização de dispositivos móveis e expansão do acesso à Internet tem aumentado o interesse da população por tais dispositivos. As empresas têm visto nesta questão, uma grande oportunidade para expandir os seus negócios, aumentar a sua carteira de clientes e fornecer melhores serviços por meio de aplicações móveis. No entanto, devido à grande oferta de aplicações por diversas empresas e programadores individuais, os utilizadores têm-se tornado cada vez mais exigentes e intuitivamente conscientes das questões de qualidade de software, como é o caso da usabilidade e acessibilidade. Neste contexto, percebe-se que uma ferramenta que permita a automatização de tais testes, reduzindo ou até mesmo eliminando completamente a execução de processos manuais, permitirá aos programadores de aplicações, atestar cada vez mais a usabilidade e acessibilidade dos sistemas de software que desenvolvem.

Visto que, o desenvolvimento de software é um processo dinâmico e os requisitos mudam ao longo do tempo, a ferramenta desenvolvida permite que os testes de usabilidade possam ser realizados à medida que o software é modificado. Desta forma, é exequível detetar problemas que possam ter surgido com alterações realizadas no software. Considera-se que o desafio proposto pela empresa Altran ao grupo de investigação REALEASE foi satisfeito. Ou seja, foi desenvolvida a ferramenta pretendida de acordo com os requisitos estipulados pela Altran, os quais todos foram atendidos.

Com o objetivo de filtrar as normas de acessibilidade Web que seria adaptáveis aos testes de aplicações móveis, estabeleceu-se as relações entre normas de acessibilidade, o que, se revelou muito útil, pois possibilitou a escolha dos Mobile Web Best Practice como base para os testes de acessibilidade.

Para além da execução automática dos testes de acessibilidade, a ferramenta permitiu integrar os resultados dos testes na ferramenta de gestão de testes Teslink, em uso na referida empresa. A arquitetura criada possibilita a integração da ATAT com outras ferramentas através de Web services. O que significa que no futuro, o serviço poderá ser disponibilizado para outros clientes, o que representa uma mais valia à Altran.

6.1 Trabalhos futuros

Como trabalhos futuros propõe-se a paralelização da execução dos testes para redução do tempo total de execução. Nos resultados obtidos, de um total de 24 test suites, cada um levou em média 37 segundos para ser completado, totalizando em média 888 segundos. Com a paralelização, pode-se então executar cada test suite de forma distribuída, possibilitando uma redução drástica do tempo total de execução dos testes. A ferramenta proposta apenas testa

Testes automáticos de acessibilidade em aplicações móveis

acessibilidade em Web apps, portanto a sua expansão para aplicações nativas representaria uma excelente oportunidade de melhorar o serviço e torna-la mais robusta. Uma das formas de obter a universalidade da informação é garantir que a mesma informação é acedida de igual forma por qualquer dispositivo móvel, neste sentido, propomos juntar aos testes de acessibilidade, os testes de compatibilidade das aplicações.

Bibliografia

- [AFT⁺12] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De, Università Federico, and I I Napoli. Using GUI Ripping for Automated Testing of Android Applications. *Proceedings of the 27th IEEE international conference on Automated Software Engineering*, pages 258-261, 2012. Available from: <http://www.cs.umd.edu/~atif/papers/AmalfitanoASE2012-abstract.html>. 17
- [ANHY12] Saswat Anand, Mayur Naik, Mary Jean Harrold, and Hongseok Yang. Automated concolic testing of smartphone apps. *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12*, page 1, 2012. Available from: <http://doi.acm.org/10.1145/2393596.2393666>\delimeter"026E30F\$nh<http://dl.acm.org/citation.cfm?doid=2393596.2393666>. 17
- [Ans90] Ansi/leee Std 829-1983. *{IEEE} {S}tandard for {S}oftware {T}est {D}ocumentation*, volume 1990. 1990. 5, 12
- [App11] Computer Applications. Mobile Software Testing - Automated Test Case Design Strategies. *International Journal*, 3(4):1450-1461, 2011. 14
- [AZAS10] K R Aida-Zade, C Ardil, and a M Sharifova. The main principles of text-to-speech synthesis system. *International Journal of Signal Processing*, 6(1):13-19, 2010. Available from: <http://www.waset.org/journals/ijice/v6/v6-1-3.pdf>. 29
- [Bal07] Barbara Ballard. *Designing the mobile user experience*. 2007. Available from: <http://books.google.com/books?hl=en&lr=&id=2n6Xi2JY6KYC&oi=fnd&pg=PR5&q=Designing+the+Mobile+User+Experience&ots=U-ORhL9qqe&sig=JPcdMjCptc76IHuSzNClkK8fdAc>. 14
- [BBC⁺10] Marco Billi, Laura Burzagli, Tiziana Catarci, Giuseppe Santucci, Enrico Bertini, Francesco Gabbanini, and Enrico Palchetti. A unified methodology for the evaluation of accessibility and usability of mobile applications. *Universal Access in the Information Society*, 9(4):337-356, 2010. xi, 18
- [Bec99] Kent Beck. *Extreme Programming Explained: Embrace Change*. 1999. 16
- [Ben05] Je Bentley. Software Testing Fundamentals — Concepts , Roles , and Terminology. *Proceedings of SAS Conference*, pages 1-12, 2005. Available from: <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Software+Testing+Fundamentals+?+Concepts+,+Roles+,+and+Terminology#0>. 11
- [BLC10] R Bandeira, R Lopes, and L Carriço. Towards mobile web accessibility evaluation. *Free and Open Source . . .*, 2010. Available from: http://www.accessible-project.eu/tl_files/documents/RBandeira-ETAPS-2010.pdf. 24
- [Boe84] Barry W. Boehm. Software Engineering Economics. *IEEE Transactions on Software Engineering*, SE-10(1), 1984. 33
- [Bog05] Robert Bogue. Use S.M.A.R.T. goals to launch management by objectives plan, 2005. Available from: <http://www.techrepublic.com/article/use-smart-goals-to-launch-management-by-objectives-plan/>. 34

- [Bra11] Giorgio Brajnik. The troubled path of accessibility engineering. *ACM SIGACCESS Accessibility and Computing*, (100):1-11, 2011. 17
- [Bud15] Raluca Budiu. Mobile User Experience: Limitations and Strengths, 2015. Available from: <http://www.nngroup.com/articles/mobile-ux/>. 15
- [Bur02] *Practical Software Testing*. Springer, 2002. 7, 8, 9, 10, 18
- [Cen14] UDC Universal Design Center. Mobile Web Best Practices (MWBP) vs. WCAG. Technical Report 818, 2014. xiii, 24, 25
- [CF14] Clearwater CF. Mobile Report, 2014. Available from: www.clearwatercf.com/documents/library/Mobile_Report_FINAL.pdf). 1
- [CMV⁺13] Gerardo Canfora, Francesco Mercaldo, Corrado Aaron Visaggio, Mauro D’Angelo, Antonio Furno, and Carminantonio Manganelli. A case study of automating user experience-oriented performance testing on smartphones. *Proceedings - IEEE 6th International Conference on Software Testing, Verification and Validation, ICST 2013*, pages 66-69, 2013. 45
- [CYM10] Tsung-hsiang Chang, Tom Yeh, and Robert C Miller. GUI Testing Using Computer Vision. (Figure 1), 2010. 17
- [CYZC] Guitao Cao, Jie Yang, Qing Zhou, and Weiting Chen. Software Testing Strategy for Mobile Phone. xi, 9, 10, 11, 12
- [DAS01] Anind Dey, Gregory Abowd, and Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, 16(2):97-166, 2001. 14
- [Day14] Patrick Day. n-Tiered Test Automation Architecture for Agile Software Systems. *Procedia Computer Science*, 28:332-339, 2014. Available from: <http://www.sciencedirect.com/science/article/pii/S1877050914001045>. 45
- [DBM13] José Manuel Díaz-Bossini and Lourdes Moreno. Accessibility to mobile interfaces for older people. *Procedia Computer Science*, 27(Dsai 2013):57-66, 2013. Available from: <http://dx.doi.org/10.1016/j.procs.2014.02.008>. 27
- [dSV10] Alberto Manuel Rodrigues da Silva and Carlos Alberto Escalreira Videira. *UML, Metodologias e Ferramentas CASE*. 2ª edition, 2010. xi, 16, 32, 33, 37, 40, 42, 43
- [Gab10] Jonathan Lazar Gabriele Meiselwitz, Brian Wentz. *Universal Usability: Past, Present, and Future*. Now Publishers, 2010. Available from: https://books.google.pt/books?id=cV0epsdrLjIC&pg=PA105&lpg=PA105&dq=Understanding+mobile+phone+requirements+for+young+adults+with+cognitive+disabilities&source=bl&ots=PF9hVNHRGW&sig=k24V5S6Q_mBjRHVD4Z-YdnYialI&hl=pt-PT&sa=X&ei=WHZsVYerIMbXyQP_hoLYBg&ved=0. 27
- [GM13] J K Geetha and M Monika. Web Application Testing : A Survey. *International Journal of Recent Technology and Engineering (IJRTE)*, 1(6):69-71, 2013. 17
- [Goo] Google. Making Applications Accessible. Available from: <http://developer.android.com/guide/topics/ui/accessibility/apps.html#label-ui>. 29

- [GWG03] Bobby George, Laurie Williams, and Williams L George B. An initial investigation of test driven development in industry. *Proceedings of the ACM Symposium on Applied Computing*, pages 1135-1139, 2003. Available from: <http://doi.acm.org/10.1145/952532.952753>~~http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.5570&rank=4~~~~http://scholar.google.ca/scholar?start=200&q=allintitle:++("test+driven"+OR+"test+first"+OR+tdd)+(development+OR+design+OR+programming+OR+app.~~ 16
- [Hen12] Shawn Lawton Henry. WAI, 2012. Available from: <http://www.w3.org/WAI/intro/wcag>. 19, 20
- [HP14] HP. Emulators vs. Real Devices for Mobile Application Testing - The Answer Might Surprise You, 2014. Available from: <http://h30499.www3.hp.com/t5/The-Future-of-Testing-Blog/Emulators-vs-Real-Devices-for-Mobile-Application-Testing-The/ba-p/5506993#.VeSn6rSpXjI>. 15
- [IAS12] K İnçki, I Ari, and H Sozer. A Survey of Software Testing in the Cloud. *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*, pages 18-23, 2012. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6258440>. 15
- [IKK14] Venkata N Inukollu, Divya D Keshamoni, and Taeghyun Kang. Factors influencing quality of mobile apps. 5(5), 2014. 1, 14
- [Inc] Apple Inc. Understanding Accessibility on iOS. Available from: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/iPhoneAccessibility/Accessibility_on_iPhone/Accessibility_on_iPhone.html#//apple_ref/doc/uid/TP40008785-CH100-SW7. 29
- [Jal08] Pankaj Jalote. *A Concise Introduction to Software Engineering*. Springer, 2008. xi, 33, 34
- [Jen08] Nick Jenkins. A Software Testing Primer. 2008. 11, 12
- [JLG07] Bo Jiang, Xiang Long, and Xiaopeng Gao. MobileTest: A tool supporting automatic black box test for software on smart mobile devices. *Proceedings - International Conference on Software Engineering*, pages 0-6, 2007. 17
- [Jov08a] M Jovanovic. Software Testing Methods and Techniques. pages 30-41, 2008. 7, 9
- [Jov08b] M Jovanovic. Software Testing Methods and Techniques. pages 30-41, 2008. 7, 9, 10
- [JT] Wayne Dick Jewett and Tom. Mapping Section 508 to WCAG 2.0. Available from: <http://www.tomjewett.com/accessibility/508-WCAG2.html>. 27
- [Kam14] Author Kelvin Kam. Mobile Emulators vs . Real Devices. Technical report, 2014. Available from: <http://www.qualitestgroup.com/assets/Mobile-emulators-vs-real-devices.pdf>. 15
- [Kan] Cem Kaner. Tutorial on Impossibility. *Law of Software Quality Column*, pages 1-16. 13
- [Kan03] Cem Kaner. What Is a Good Test Case? *STAR East*, pages 1-16, 2003. 6

- [Key15] Keynote. Testing Strategies and Tactics for Mobile Applications. Technical report, 2015. Available from: <http://www.keynote.com/resources/white-papers/testing-strategies-tactics-for-mobile-applications>. 15
- [KK13] B. Kirubakaran and V. Karthikeyani. Mobile application testing – Challenges and solution approach through automation. *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*, pages 79-84, 2013. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6496451>. 1
- [KNR09] Eun Ha Kim, Jong Chae Na, and Seok Moon Ryoo. Test automation framework for implementing continuous integration. *ITNG 2009 - 6th International Conference on Information Technology: New Generations*, pages 784-789, 2009. 13
- [Kon96] The Government of Hong Kong. Mobile Application Accessibility Handbook. 1996. 27
- [Lat14] Sohaib (University of Gujrat) Latif. Both are different ways of requirement gathering methods., 2014. Available from: http://www.researchgate.net/post/What_are_the_major_differences_in_functional_and_nonfunctional_requirements. 34
- [LIO96] J. L. LIONS. ARIANE 5: Flight 501 Failure, 1996. Available from: <https://www.ima.umn.edu/~arnold/disasters/ariane5rep.html>. 6
- [LK05] Inseong Lee and Jinwoo Kim. Use contexts for the mobile internet: a longitudinal study monitoring actual use of mobile internet services. *International Journal of Human-Computer ...*, 18(3):269-292, 2005. Available from: http://www.tandfonline.com/doi/abs/10.1207/s15327590ijhc1803_2. 14
- [LXC12] Yepang Liu, Chang Xu, and S C Cheung. Verifying Android Applications Using Java PathFinder State Key Laboratory for Novel Software Technology Table of Contents. 2012. 17
- [Mar] Robert C Martin. *Clean Code Collection*. 61
- [Mar00] Brian Marick. When Should a Test Be Automated? pages 1-10, 2000. Available from: <http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=ART&ObjectId=2010#authorbio>. 29
- [MEK⁺12] Riyadh Mahmood, Naeem Esfahani, Thabet Kacem, Nariman Mirzaei, Sam Malek, and Angelos Stavrou. A whitebox approach for automated security testing of Android applications on the cloud. *2012 7th International Workshop on Automation of Software Test (AST)*, pages 22-28, 2012. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6228986>. 17
- [MM12] Lourdes Moreno and Paloma Martínez. A Review of Accessibility Requirements in Elderly Users' Interactions with Web Applications. *Proceedings of the 13th International Conference on InteracciÓN Persona-Ordenador*, 17:47:1--47:2, 2012. Available from: <http://doi.acm.org/10.1145/2379636.2379682>. 28
- [MO08] Mohammad Mannan and P C Van Oorschot. Security and Usability : The Gap in Real-World Online Banking. *Proceeding NSPW '07 Proceedings of the 2007 Workshop on New Security Paradigms*, pages 1-14, 2008. 10

- [MRH07] Tafline Murnane, Karl Reed, and Richard Hall. On the learnability of two representations of equivalence partitioning and boundary value analysis. *Proceedings of the Australian Software Engineering Conference, ASWEC*, pages 274-283, 2007. 9
- [MSD⁺11] Rabeb Mizouni, M. Adel Serhani, Rachida Dssouli, Abdelghani Benharref, and Ikbale Taleb. Performance Evaluation of Mobile Web Services. *2011 IEEE Ninth European Conference on Web Services*, pages 184-191, 2011. 17
- [MSMM14] B A L Madhushani, P H A M De Silva, W A L Madushanka, and M G T H Malalagama. Challenges in Mobile Application Testing : Sri Lankan Perspective. 3(X), 2014. 15
- [Muc12] Henry Muccini. Software testing of mobile applications: Challenges and future research directions. *Automation of Software Test (AST), 2012 7th International Workshop on*, pages 29-35, 2012. Available from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6228987. 12, 13
- [Nid12] Srinivas Nidhra. Black Box and White Box Testing Techniques - A Literature Review. *International Journal of Embedded Systems and Applications*, 2(2):29-50, 2012. 9, 10
- [Nie93] Jakob Nielsen. Usability Engineering. page 340, 1993. 17
- [Pap13] White Paper. Quality of Service (QoS) and Policy Management in Mobile Data Networks. (December):1-22, 2013. 17
- [Pat12] Ron Patton. *Software Testing (2nd Edition)*. 2012. 6, 8, 16
- [PDKK02] Barbara Paech, AH Dutoit, Daniel Kerkow, and Antje Von Knethen. Functional requirements, non-functional requirements, and architecture should not be separated-A position paper. *Design*, 2002. Available from: <https://www.bruegge.in.tum.de/static/publications/pdf/106/paech2002nfr.pdf>. 34
- [PK07] H. Petrie and O. Kheir. The relationship between accessibility and usability of websites. pages 397-406, 2007. Available from: <http://dx.doi.org/10.1145/1240624.1240688>. 17
- [Pre05] Roger S. Pressman. *Software Engineering*. 2005. 31
- [Qua10] S M K Quadri. Software Testing - Goals , Principles , and Limitations. 6(9):7-10, 2010. xi, 8, 9
- [Rab08] Jo Rabin. Mobile Web Best Practices 1.0, 2008. Available from: <http://www.w3.org/TR/mobile-bp/>. 24
- [Reh] US Access Board. Available from: <http://www.access-board.gov/the-board/laws/rehabilitation-act-of-1973>. 25
- [Reh98] Rehabilitation Act of 1973. US Government, 1998. Available from: <http://www.section508.gov/section-508-standards-guide>. 26
- [Roy70] W W Royce. Managing the Development of Large Software Systems. *IEEE WESCON (Reprinted in Proceedings Ninth International Conference on Software Engineering.)*, (August):1-9, 1970. 16

- [RS07] Doug Rosenberg and Matt Stephens. *Use Case Driven Object Modeling with UML Theory and Practice*. 2007. 31
- [Sat04] I Satoh. Software Testing for Wireless Mobile Computing. (October):58-64, 2004. 17
- [SFKS03] Speech-to-text Summarization, Sadaoki Furui, Tomonori Kikuchi, and Yousuke Shinaka. Speech-to-Speech and Speech-to-Text Summarization. pages 100-106, 2003. 29
- [Sha] Shawn Lawton Henry. Introduction to Web Accessibility. Available from: <http://www.w3.org/WAI/intro/accessibility.php>. 19
- [Sha14] R M Sharma. Quantitative Analysis of Automation and Manual Testing. *International Journal of Engineering and Innovative Technology (IJEIT)*, 4(1):252-257, 2014. 13
- [Sik11] Leslie Sikos. *Web Standards: Mastering HTML5, CSS3, and XML*. 2011. Available from: http://books.google.com/books?hl=en&lr=&id=5czpyb45uNYC&oi=fnd&pg=PR1&dq=Web+Standards+-+Mastering+HTML5,+CSS3+and+XML&ots=1MxFu09fUi&sig=IYlsjk1UfiehVjr_hyvyrVcpDWU. 21, 22, 27
- [Sma15] Smartbear. Automated Testing in Agile Environments, 2015. Available from: <https://support.smartbear.com/articles/testcomplete/automated-testing-agile-environment/>. 13
- [SMT⁺10] Forrest Shull, Grigori Melnik, Burak Turhan, Lucas Layman, Madeline Diep, and Hakan Erdogmus. What do we know about test-driven development? *IEEE Software*, 27(6):16-19, 2010. 16
- [Som10] Ian Sommerville. *Software Engineering*. 2010. xi, 5, 33
- [Tas02] G. Tassey. The economic impacts of inadequate infrastructure for software testing. *Quality*, page 309, 2002. Available from: <http://www.nist.gov/director/prog-ofc/report02-3.pdf>. 6
- [Tra99] Eushiuan Tran. Verification / Validation / Certification Abstract : Contents :. *Spring*, 1999. 8
- [Tra13] David Travis. Adapting your usability testing practise for mobile, 2013. Available from: <http://www.userfocus.co.uk/articles/testing-for-mobile.html>. 15
- [Ute15] Utest. Crowdsourcing your mobile app testing, 2015. Available from: http://www.utest.com/ext-v4/g/crowdsourcing-your-mobile-app-testing?ls=BannerAd&cc=Pa&mc=Display-SQE-Mobile_Testing-CPA-Oct2012. 1
- [W3C] W3C. WCAG 1.0. Available from: <http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505/>. 21
- [W3C08] W3C. WCAG, 2008. Available from: <http://www.w3.org/WAI/intro/wcag>. xi, 23
- [W3C14] W3C. HTML 5 - A vocabulary and associated APIs for HTML and XHTML, 2014. Available from: <http://www.w3.org/TR/html5/obsolete.html>. 19, 59
- [Web] WebAim. Available from: <http://webaim.org/articles/laws/usa/rehab#intro>. 26
- [WF10] Anthony I Wasserman and Fosser. Software Engineering Issues for Mobile Application Development. *FoSER*, pages 1-4, 2010. 13, 14

Testes automáticos de acessibilidade em aplicações móveis

- [Wil01] Laura Williams. White-Box Testing. *Whit Box Testing*, 2001. 10
- [Wil06] Laurie Williams. Testing Overview and Black-Box Testing Techniques. page 26, 2006. Available from: <http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf>. 9, 11, 12
- [Wor15] World Stats. Internet World Stats, 2015. Available from: <Http://www.internetworldstats.com/stats.htm>. 1
- [ZA05] Dongsong Zhang and Boonlit Adipat. Challenges, Methodologies, and Issues in the Usability Testing of Mobile Applications. *International Journal of Human-Computer Interaction*, 18(3):293-308, 2005. 14

Apêndice A

Anexos

A.1 Datasheets dos componentes utilizados

