Ferramentas de Teste de Sistema Open Source: Comparação e Caracterização

Diana Costa[2], Marco Silva[2], Miguel Brito[1], Pedro Ribeiro[1], and Vitor Peixoto[2]

[1] Centro de Investigação Algoritmi, Universidade do Minho, Campus de Gualtar, Braga 4710, Portugal
[2] Universidade do Minho, Campus de Gualtar, Braga 4710, Portugal

Resumo No desenvolvimento de *software* é típico que, em primeira instância, ocorra o desenvolvimento do cógigo, onde os programadores apenas realizam testes unitários, e só depois se realizem os testes pela equipa de *testers*. Os erros que fluem da fase de desenvolvimento para a fase de teste podem ser caros de corrigir por serem tardiamente detetados, ou não são detetados de todo, o que pode ser especialmente perigoso em sistemas críticos, levando desde a falência de empresas até à perda de vidas.
Assim, há uma necessidade em detetar *bugs* o mais cedo possível. Cada vez mais as IDEs oferecem a possibilidade de integração de ferramentas de teste, o que é importante na medida em que vários erros já vão corrigidos para a fase de teste, dando azo a que os *testers* se possam focar e detetar outras imperfeições.

Keywords: Gestão do Processo de Software, IDE, Open Source, Teste, Selenium, TestComplete, TestingWhiz, Telerik Test Studio, UI Vision, UFT

1 Introdução e Contextualização

Um projeto de desenvolvimento de sistemas de informação possui várias fases, onde a utilização de métodos e ferramentas podem auxiliar o seu bom desenvolvimento e progressão. Porém, mesmo com a utilização desses recursos, podem existir erros no produto gerado, já que esta atividade envolve uma série de etapas em que as oportunidades de injeção de falhas são muito grandes (falhas na arquitetura, desenho, requisitos, análise, implementação, ...). Esses erros podem levar o projeto a ter muitas dificuldades, elevar os custos ou até mesmo levar ao seu insucesso. Desta forma, o desenvolvimento deve ser acompanhado de uma série de testes para estabelecer uma garantia de qualidade. Entre as diversas fases do processo de teste de *software*, está contemplado o teste de sistema, que será mais abordado ao longo do presente relatório.

Assim, a utilização de ferramentas de testes de sistema é uma forte aliada do processo de desenvolvimento de sistemas de informação. No entanto, existe uma dificuldade em encontrar a ferramenta que corresponda, de forma mais

significante e abrangente, às necessidades gerais do projeto, tendo em vista que cada projeto tem suas particularidades. Em virtude do exposto, definiu-se como questão principal da pesquisa:

"Quais as principais ferramentas open source que mais auxiliam o desenvolvimento de sistemas de informação no processo de teste de software?"

Com isto em mente, este ensaio tem como objetivo principal a descrição e comparação das diversas ferramentas de teste de sistema existentes, e em que medida estas permitem validar o *software* desenvolvido através da integração com IDEs. Desta forma, nas próximas secções, começa-se por fazer uma introdução teórica de testes de sistema, assim como indicar as ferramentas escolhidas pelo grupo para análise e comparação. Por fim, será apresentado o plano de trabalho, e um quadro comparativo das ferramentas, explicando todos os parâmetros do mesmo, assim como feito um balanço da pesquisa e descobertas do grupo.

2 Teste de sistema

Existem muitas estratégias que podem ser utilizadas para testar *software*. A prática mais recorrente baseia-se numa visão incremental do teste, culminando com o teste de sistema. O teste de sistema é uma fase do processo de teste de *software* em que o sistema, já completamente integrado, é verificado quanto aos seus requisitos num ambiente de produção. Essa verificação não requer conhecimento da estrutura (lógica) interna do sistema, não se limitando a testar somente requisitos funcionais, mas também outros aspetos não-funcionais do sistema. É comum chamar este teste de caixa preta, pois o sistema é testado com tudo interligado: *batch jobs*, base de dados, serviços *web*, etc.

Estes testes são realizados pela equipa de teste de *software* e visam a execução do sistema dentro de um ambiente operacional controlado, para validar a perfeição na execução das suas funções. Na prática eles são executados em condições semelhantes àquelas que o utilizador irá utilizar, verificando, por exemplo, se os componentes são compatíveis, se eles interagem corretamente, se transferem os dados certos no momento certo, entre outros.

3 Ferramentas

Para agilizar os testes de *software* e minimizar o tempo despendido nessas tarefas, o surgimento de ferramentas que permitam automaticamente efetuar esses testes desempenha um papel muito importante.

Atualmente, tem-se uma panóplia de ferramentas de fácil acesso e que permitem agilizar o processo de teste de *software*. Abaixo serão escrutinadas algumas das ferramentas mais importantes e as suas caraterísticas mais relevantes. Todas as ferramentas selecionadas tiveram por base o **teste de sistema** e a **possibilidade de integração com IDEs**. No entanto, as ferramentas de teste vão desde os testes de carga/stress, a testes de unidade, entre outros.

3.1 Ferramentas escolhidas

Selenium Esta é, sem dúvida, uma das principais ferramentas de teste para novos *softwares*. O Selenium permite a criação de testes funcionais para web, e os testes são executados diretamente em qualquer *browser*, desde que suporte *JavaScript*. É possível avaliar a performance de uma aplicação em diferentes navegadores e plataformas, como *Windows*, *MacOS* e *Linux*. Além disso, é compatível com muitas linguagens, como Java, PHP, C#, Python, Groovy, Ruby, entre outras. Esta ferramenta está dividida nos seguintes componentes:

- **Selenium Core**: O teste é feito diretamente pelo *browser*;
- **Selenium IDE**: é uma ferramenta criada com um *plug-in* do navegador *Firefox*, e permite gravar o que é feito nesse *browser* num *script* cujo conteúdo é disponibilizado em forma de tabela, sendo também possível repetir, se necessário, as ações gravadas no navegador;
- **Selenium RC**: são feitos testes automáticos e usada qualquer linguagem de programação. É possível alterar o que foi gravado acrescentando *loops*, funções e bibliotecas que venham a ser criados;
- **Selenium Grid**: permite que os testes sejam distribuídos em diversos computadores e executados de forma coordenada.

Atualmente, a aplicação passou a contar com uma API *WebDriver*. Com isto, é possível conduzir avaliações num navegador de forma nativa como um utilizador, ou numa máquina remota usando o servidor Selenium.

Unified Functional Testing O Unified Functional Testing é uma ferramenta, lançada em 2001 sob o nome de QuickTest Professional, propriedade da Hewlett Packard. Desde 2016 que é desenvolvida pela Micro Focus. Trata-se então de uma das referências do mercado com mais de 18 anos desde o primeiro lançamento.

Especializada em soluções para automação de testes de aplicações e ambientes de *software*. Executa testes através de uma interface nativa ou web, identificando os objetos na interface da aplicação e executando as operações desejadas (cliques, eventos de teclado, etc.). Pode também capturar propriedades dos objetos.

Devido à prolongada existência desta ferramenta no mercado, muitas das suas funcionalidades dependem ainda de ferramentas algo obsoletas, como *ActiveX* ou o uso de uma linguagem de *scripting* não orientada aos objetos, *VBScript*. Assim sendo, esta ferramenta está apenas disponível para *Microsoft Windows* e não é compatível com alguns *browsers*, sendo o nome mais sonante o do *Opera*.

Como vantagens apresenta a larga integração com alguns IDEs e a possibilidade de gerar testes *data-driven* através do acesso a uma base ou ficheiros de dados.

Como desvantagens, destaca-se algumas das tecnologias obsoletas que ainda são empregues nesta ferramenta e que reduzem a sua disponibilidade a diversos setores do mercado. Para além disso, o elevado custo da licença por vezes obriga a que a equipa de testes seja reduzida, por fatores orçamentais. Isto leva a que a fase de testes seja separada da fase de desenvolvimento, ao invés de um esforço contínuo de colaboração, contradizendo as técnicas de teste *Agile*.

UI Vision Kantu UI Vision é uma ferramenta de automação de tarefas que utilizando inteligência artificial combina o reconhecimento de texto e a imagem do estado da arte para a execução de ações sem a intervenção humana. Esta ferramenta disponibiliza diversas componentes a aplicar para automação de tarefas em diferentes tecnologias.

Uma das grandes preocupações atualmente é a aceleração do processo de desenvolvimento de um determinado produto de software. Até à sua conclusão, este terá de ser testado por forma a garantir que se encontra desenvolvido segundo as especificações do cliente. Desta forma, a execução de testes durante a fase de desenvolvimento permite a identificação mais cedo de possíveis erros e assim facilitando o processo final de testes.

Para isso, esta ferramenta oferece a componente visual de automação web (Visual Web Automation) que permite a automação das tarefas de uma forma visual, fácil e intuitiva. Todas as ações realizadas por um utilizador, quer seja o preenchimento de formulários, introdução de dados de autenticação ou até mesmo cliques em determinados elementos da interface podem ser capturados e futuramente replicados, permitindo assim a verificação da correção das funcionalidades a serem implementadas no momento.

Com uma breve pesquisa, é possível perceber que existem diversas ferramentas que são capazes de automatizar tarefas apenas em interfaces web e não em aplicações *desktop*. Por forma a colmatar esta lacuna, a *UI Vision* disponibiliza a componente *UI Vision for Desktop Automation*, capaz de automatizar tarefas realizadas por um utilizador em aplicações *desktop*. Esta componente utiliza tecnologias como OCR (optical character recognition) para a identificação de imagens e texto no ecrã apresentado ao utilizador. Para além da compatibilidade com aplicações *Windows*, *Mac* e *Linux*, a automação pode ser entendida também a emuladores *Android*, o que poderá ser bastante útil. A automação pode ser utilizada ainda para a automação da inserção de comandos em linhas de comando ou até acesso a *desktop* remotos. Resumidamente, qualquer elemen-

tos que apareça no ecrã no utilizador e com que este possa interagir, pode ser replicado utilizando esta componente.

Finalmente, a *UI Vision* disponibiliza ainda a integração com a ferramenta *Selenium IDE*, possibilitando assim a implementação de plataformas de teste híbridas. Para os utilizadores mais puristas, é possível ainda utilizar todas as funcionalidades a partir da linha de comandos ou ser integrado nas mais populares linguagens de programação (Python, Powershell, etc).

TestComplete TestComplete é uma plataforma funcional (desenvolvida pela AutomatedQA) que oferece diversas soluções para automação de testes de software para desktop, web e aplicações mobile. Algumas das funcionalidades oferecidas pela plataforma são:

- Teste GUI;
- Suporte para *scripts* nas linguagens: JavaScript, Python, VBScript, JScript, DelphiScript, C ++ Script e C# Script;
- Visualizador de teste;
- Avaliação de *scripts*;
- Teste de gravação e reprodução.

Com representantes em varias partes do mundo, a TestComplete oferece o mesmo conjunto de funcionalidades que os seus principais concorrentes (IBM Rational Functional Tester, Borland SilkTest e Unified Functional Testing). Por se tratar de uma ferramenta de baixo custo, hoje muitas empresas, inclusive as de grande porte, estão a migrar ou a adquirir esta ferramenta, tendo em conta a facilidade de trabalhar com a mesma.

Para além do já mencionado, um dos destaques da TestComplete é o suporte nativo para a automação de sistema desenvolvidos em várias tecnologias, tais como: .NET, Java, Visual Basic, Borland Delphi, WEB, PowerBuilder, FoxPro, entre outros. Além disso, o TestComplete oferece recursos para a realização de testes de desempenho para aplicações WEB e testes funcionais de WEB Services.

TestingWhiz A Testing Whiz é uma ferramenta de teste de software bastante utilizada, que fornece soluções de teste automatizadas. Possui diversos pacotes: a edição Enterprise, por exemplo, tem soluções como testes de bases de dados, de API, web, nuvem, de aplicações para dispositivos móveis, entre outros.

A plataforma possui ainda outros diferenciais: é possível fazer testes baseados em palavras-passe, em dados e baseados em risco. Uma outra característica importante é a integração com ferramentas, estruturas e plataformas populares, que garantem uma automação contínua e a execução de casos de teste em diferentes ambientes e condições. Com essas integrações, o TestingWhiz permite que sejam direcionadas as melhores conexões através da arquitetura técnica da aplicação e que se expandam os horizontes de automação. Algumas das integrações significativas que foram disponibilizadas no TestingWhiz para fornecer uma experiência de automação eficaz para os engenheiros de teste e gerentes de QA encontram-se abaixo descritas:

- Ferramentas de gestão de testes;
- Ferramentas de integração contínua;
- Ferramenta de deteção de *bugs*;
- Bases de dados;
- Plataformas móveis.

Telerik Test Studio O Telerik TestStudio oferece uma solução para automatizar testes de aplicações, tanto de desktop como web e dispositivos móveis, incluindo avaliações de Interface de utilizador (UI), carga e desempenho, fundamentais para um software de qualidade.

A plataforma oferece suporte de linguagens de programação como HTML, AJAX, ASP.NET, JavaScript, Silverlight, WPF e MVC. Além disso, oferece integração com o Visual Basic Studio 2010 e 2012. É também, possível gravar e reproduzir os procedimentos, realizar testes manuais, agendar testes automáticos e fazer a integração com ferramentas de deteção de *bugs*.

3.2 Estudo comparativo

Cada ferramenta de automação de testes está equipada com funcionalidades capazes de satisfazer as necessidades específicas de uma fatia do mercado ou com funcionalidades que abrangem várias fatias de forma mais superficial.

Deste modo, para poder estabelecer uma comparação entre as diversas ofertas existentes da melhor maneira possível, deve-se comparar as ferramentas de acordo com diversas funcionalidades existentes e métricas específicas, tendo sempre como foco principal a integração destas ferramentas com IDEs.

Com o objetivo de dividir as funcionalidades em categorias, projetaram-se três tabelas individuais:

1. **Funcionalidades gerais:** estabelecem funcionalidades básicas e praticamente comuns a todas as ferramentas.
   (a) **Geração de relatórios:** a ferramenta gera um ficheiro ou página relativos ao conjunto de testes aplicados. Estes relatórios podem incluir texto, pontuações (*scores*), gráficos, etc;
   (b) **Plataforma *web*:** a ferramenta disponibiliza uma plataforma *web* que permite consultar documentação e tutoriais para possibilitar a redução da curva de aprendizagem;
   (c) **Tipo de controlo de acesso de utilizador:** a ferramenta permite efetuar testes ao sistema, tendo em conta grupos de utilizadores diferentes;
   (d) **Integração com ferramentas *BugTrack*:** a deteção de falhas e erros é permitida através da integração de ferramentas específicas como o *JIRA* ou o *Lighthouse*;
   (e) **Histórico de testes executados:** a ferramenta implementa automaticamente um registo histórico dos testes efetuados;
   (f) **Especificação de requisitos:** é possível efetuar testes baseados em requisitos específicos do sistema;

   (g) **Importar/exportar conteúdos:** a ferramenta permite efetuar *data-driven tests* baseados em dados carregados de ficheiros CSV, XML, etc. O oposto também é válido para a exportação;
   (h) **Suporte a bases de dados:** a ferramenta permite efetuar *data-driven tests* baseados em dados existentes numa base de dados;
   (i) **Funções para automação de teste:** a ferramenta possibilita a execução automática de testes, através de, por exemplo, criação de *scripts*.
2. **Usabilidade:** Alguns parâmetros de usabilidade que medem a facilidade de uso da ferramenta e aparência.
   (a) **Muitos menus:** interface assoberbada de menus, mais do que típicos menus superiores (*ribbon*), o que não quer dizer que estes não sejam intuitivos;
   (b) **Letra pequena:** a utilização de letra pequena pode dificultar a utilização da ferramenta, mas carateres gigantes provocam uma interface confusa e com pouco espaço para funcionalidades;
   (c) **Cliques para aceder funcionalidades:** se uma funcionalidade necessita de um elevado número de cliques para ser alcançada e executada (por exemplo, mais de 15), considera-se que a organização de menus não é a mais eficiente e atrasa o processo de teste;
   (d) **Recuperação rápida de erros de utilizador:** capacidade de retroceder e avisos sobre possíveis futuros passo;.
   (e) **Documentação:** a ferramenta tem uma secção de ajuda que inclui documentação, tutorias e links para *websites*;
   (f) **Ferramenta rápida:** a ferramenta executa testes a uma velocidade razoável num ambiente com os requisitos recomendados;
   (g) **Utilização de recursos:** esforço computacional que a ferramenta exige para a execução de testes. Definidos através dos requisitos recomendados da ferramenta;
   (h) **Curva de aprendizagem:** opinião do grupo relativamente à curva de aprendizagem da ferramenta após 6 horas de utilização. A curva de aprendizagem está associada à facilidade de uso da interface e da necessidade, ou não, de escrita de código (*scripting*, integração de bases de dados, etc.). Esta característica é propositadamente e inevitavelmente subjetiva.
3. **Integração com IDEs**: características sobre a ferramenta em si e acerca da integração e acesso à integração.
   (a) **Facilidade de integração:** métrica relativa ao número de cliques (por exemplo, mais do que 15) ou necessidade de escrita de código para a integração da ferramenta com um IDE;
   (b) **Relatórios completos/legíveis:** os relatórios gerados permitem fornecer informação útil para a constante melhoria e correção de erros do sistema, de acordo com o método *Agile*;
   (c) **Integração contínua:** a ferramenta permite a integração contínua, através de ferramentas como Jenkins ou Bamboo;
   (d) **Linguagem de *scripting*:** a ferramenta suporta a criação de *scripts* em diversas linguagens de programação, idealmente mais do que uma;

(e) **Paralelização de testes:** a ferramenta permite a execução de vários testes, normalmente num ambiente distribuído;
(f) ***Job Scheduler***: a ferramenta permite o agendamento de um determinado conjunto de testes definido pelo utilizador;
(g) **Presença de Inteligência Artificial:** a ferramenta utiliza inteligência artificial em qualquer aspeto, como para otimizar o conjunto de testes efetuados ao sistema;
(h) **Integração com vários IDEs:** a integração é possível com diferentes IDEs, e não apenas com um;
(i) **Fornece cenários de recuperação:** a ferramenta oferece cenários alternativos em caso de falhas na execução de testes;
(j) **Curva de aprendizagem:** a integração da ferramenta com um IDE é simples, intuitiva e não envolve a necessidade de escrita de código. Esta característica é propositadamente e inevitavelmente subjetiva.

É de notar que foi feito um esforço de modo a que todos estes parâmetros fossem objetivos e incontestáveis. No entanto, dada a natureza de alguns (como "Curva de aprendizagem"ou "Ferramenta rápida"), tal foi impossível e a opinião poderá divergir de acordo com o utilizador e com a sua experiência na área.

Funcionalidades Gerais - Quadro Apresenta-se, de seguida, o quadro final relativo às funcionalidades gerais das ferramentas analisadas. É de notar que, no que toca à ferramenta *UI Vision* e *Selenium*, na característica "Histórico de Testes Executados", foi considerado que não cumpria esta qualidade uma vez que é da responsabilidade do utilizador guardar os resultados da execução dos testes ao longo do tempo.

	UI Vision	Selenium	UFT	TestComplete	TestingWhiz	Telerik Test Studio
Geração de Relatórios	✓	✓	✓	✓	✓	✓
Plataforma Web	✓	✓	✓	✓	✓	✓
Tipo de Controlo de Acesso de Utilizador	✗	✗	✓	✓	✓	✓
Integração com Ferramentas BugTrack	✗	✗	✓	✓	✓	✓
Histórico de Testes Executados	✗	✗	✓	✓	✓	✓
Especificação de Requisitos	✗	✗	✓	✓	✗	✗
Imp/Exportação de conteúdos	✓	✓	✓	✓	✓	✓
Suporte a Bases de Dados	✓	✓	✓	✓	✓	✓
Funções para automação de teste	✓	✓	✓	✓	✓	✓

Figura 1: Tabela de Funcionalidades Gerais

Critérios de Usabilidade - Quadro Mostra-se, em baixo, o quadro final relativo às caraterísticas de usabilidade das ferramentas escolhidas pelo grupo para análise. É importante notar que, no que toca ao atributo "Utilização Elevada de Recursos", as interfaces gráficas das ferramentas *UI Vision* e *Selenium* são executadas no contexto de um navegador web (formato *plugin*). Para além destas interfaces, são disponibilizadas também *API's* que permitem a execução de testes de sistema de uma forma programática. Assim, os requisitos essenciais da máquina na qual os testes irão ser executados são os requisitos da plataforma

de suporte à execução dos mesmos, mais propriamente, o navegador em questão ou a linguagem utilizada. Adicionalmente, considerou-se que uma elevada utilização de recursos baseia-se no facto de a ferramenta necessitar de 8Gb ou mais de memória RAM e/ou 15Gb ou mais de disco, tendo em conta também as especificações do processador.

	UI Vision	Selenium	UFT	TestComplete	TestingWhiz	Telerik Test Studio
Muitos Menus	✗	✗	✓	✓	✓	✓
Letra Pequena	✗	✗	✓	✓	✗	✗
Nº Elevado de Cliques para Aceder a Funcionalidade	✗	✗	✓	✗	✗	✓
Recuperação Rápida de Erros de Utilizador	✓	✓	✓	✓	✓	✓
Secção 'Help'/Manual Completo/Legível	✓	✓	✓	✓	✓	✓
Ferramenta Rápida	✓	✓	✓	✓	✓	✓
Utilização Elevada de Recursos	✗	✗	✓	✓	✓	✗
Curva de Aprendizagem Alta	✓	✓	✓	✗	✗	✗

Figura 2: Tabela de Critérios de Usabilidade

Critérios de Integração com IDEs - Quadro Apresenta-se, em baixo, o quadro final relativo às caraterísticas de integração com IDEs das ferramentas analisadas. É importante denotar que, quanto à característica "Facilidade de Integração", considerou-se que esta podia ser bilateral (possibilidade de integração quer a partir da ferramenta, quer a partir do IDE) ou unilateral (integração apenas num dos sentidos). Quanto ao atributo "Fornece Failure Recovery Scenarios", considerou-se que se poderiam definir previamente os casos de erro/ação, onde se especificavam ações prévias, ou o facto de, por outro lado, o sistema suspender a execução do teste até que o utilizador execute uma ação para continuar.

Relativamente à ferramenta *TestingWhiz*, esta não oferece integração direta com os ambientes de desenvolvimento mais comuns, mas sim com o *VSTS - Visual Studio Team Services* ou, segundo a nova nomenclatura, *Azure DevOps*. Posteriormente, esta ultima ferramenta é capaz de integrar com os mais comuns ambientes de desenvolvimento (*Visual Studio*, *Eclipse*, entre outros). De facto, o método de integração não é o mais simples e direto mas, por uma propriedade transitiva, e visto que, esta disponibiliza funcionalidades bastante úteis num ambiente de desenvolvimento, este mecanismo de integração foi considerado como viável.

	UI Vision	Selenium	UFT	TestComplete	TestingWhiz	Telerik Test Studio
Facilidade de Integração	✓	✓	✗	✓	✓	✓
Relatórios Completos/Legíveis	✓	✓	✓	✓	✓	✓
Integração Contínua	✗	✓	✓	✓	✓	✓
Scripts com Várias Linguagens de Programação	✓	✓	✗	✓	✓	✓
Paralelização de Testes	✗	✓	✓	✓	✓	✓
Job Scheduler	✗	✗	✓	✓	✓	✓
Presença de AI	✓	✗	✗	✓	✗	✗
Integra com vários IDEs	✗	✓	✓	✗	✓	✗
Fornece Failure Recovery Scenarios	✗	✓	✓	✓	✓	✓
Curva de Aprendizagem Alta	✓	✓	✓	✗	✗	✗

Figura 3: Tabela de Critérios de integração com IDEs

Análise Final Observando as caraterísticas e funcionalidades das ferramentas de automação de teste de sistema apresentadas no estudo efetuado, podemos passar agora a uma análise comparativa das vantagens e desvantagens no uso de cada uma destes aplicativos.

Observando a nossa primeira ferramenta, **UI Vision**, fica claro que se encontra equipada com funcionalidades inovadoras, principalmente com a integração de inteligência artificial. Uma área computacional em crescimento exponencial e cuja integração é imensamente valorizada no contexto atual da informática. Porém falha em diversos pontos fulcrais onde outras ferramentas cumprem. Apesar de permitir a integração com IDEs, esta acaba por não ser totalmente potenciada, principalmente devido à inexistência de uma integração contínua, indo contra o princípio de métodos Agile. Falha ainda em outros pontos, não relacionados com a integração com IDEs, como cenários de recuperação de falhas, ou a integração com ferramentas de *BugTrack*.

A atual ferramenta líder de mercado, **Selenium**, é apelativa pela sua interface familiar e de fácil uso, mas principalmente pela sua integração com vários IDEs, quando a maioria das ferramentas de automação de teste só permite a integração com um único IDE ou por vezes com nenhum. Outro ponto a favor desta ferramenta, é o facto de ser *open-source*, logo o esforço orçamental no desenvolvimento do projeto torna-se menor e permite ter uma equipa de teste alargada, facilitando o processo Agile.

Uma das ferramentas da "velha-guarda"é a **UFT**. Esta ferramenta já conta com 18 anos de experiência, porém encontra-se estagnada, utilizando recursos e ferramentas de gerações computacionais atrasadas. Apesar disso, tem diversas vantagens, relativamente às novas ferramentas minimalistas. A sua utilização requer a escrita de algum código, mas acaba por permitir uma integração contínua com vários IDEs, podendo correr diversos testes em paralelo, incluíndo *data-driven tests*. O seu leque alargado de funcionalidades permite à UFT ainda ter relevância no cenário atual, apesar da dificuldade de utilização imposta pela ferramenta. Como mencionado na introdução a esta ferramenta, acaba por ir em contra o processo Agile, pelos motivos contrários ao Selenium, visto esta ferramenta ter um custo efetivamente alto.

A **TestComplete** é bastante semelhante à UFT, contudo tem maior usabilidade, integração de inteligência artifical e um custo de licença reduzido, mas em contrapartida apenas permite a integração com um IDE. Apesar desta desvantagem, a TestComplete tem ganho fatia de mercado em comparação à UFT, estando mais preparada para os desafios do futuro.

A última ferramenta que permite a implementação de mais que um IDE analisada é a *TestingWhiz*. Esta ferramenta permite uma integração contínua e diversas outras funcionalidades igualmente presentes em ferramentas concorrentes, não tendo de facto uma funcionalidade onde se destaque. Acaba ainda assim por ter bastantes funcionalidades em comparação a outras ferramentas, como a UI Vision ou o Selenium, sendo por isso relevante no mercado atual.

Por fim, o **TestStudio** da Telerik, acaba por se destacar na usabilidade, apresentando uma interface agradável e familiar, devido à sua semelhança à

linha de aplicativos *Microsoft Office*. Integra grande parte das funcionalidades das restantes ferramentas, permitindo a integração com Visual Studio, apenas. Contudo, sendo este um dos IDEs mais utilizados, acaba por receber uma fatia de mercado considerável, sendo esta ferramenta aconselhável para projetos de pequena ou média dimensão.

Estabelecendo assim uma análise final às ferramentas estudadas, é finalmente possível efetuar uma escolha sobre qual utilizar no desenvolvimento de um aplicativo. Este estudo é essencial, pois a utilização da ferramenta correta, permite reduzir os custos e o tempo dispendido. Trata-se portanto de uma fase importante a ter no desenvolvimento de *software*.

4 Plano de trabalho

Ao longo do tempo, diversos projetos nas mais diversas áreas são executados, alguns com uma maior visibilidade, por exemplo, novas estradas ou pontes mas também projetos de menor visibilidade como uma obra a realizar numa determinada empresa. Independentemente da sua natureza, parte destes projetos serão um completo sucesso e outros sofrerão anomalias que podem levar ao seu fracasso. O desfecho de um projeto depende de inúmeros fatores de elevada complexidade mas, a realização da fase de planeamento com sucesso colocará certamente este mais próximo do sucesso.

A falta de planeamento de um projeto coloca definitivamente este mais longe do sucesso. Não só com uma definição clara e incisiva do objetivo do projeto, mas também com uma calendarização de todas as fases a alcançar, a probabilidade de sucesso definitivamente será mais elevada. Esta é uma tarefa aparentemente simples e básica pelo que frequentemente esta é executada de uma forma apressada e inconsciente. Outros fatores externos podem também contribuir para um mau desenvolvimento do plano do projeto, por exemplo restrições temporais, o que fará com que o planeamento realizado ao invés de conduzir ao sucesso, torne ainda mais certo o seu fracasso. Um projeto com um planeamento pobre nunca irá produzir bons resultados.

Um bom plano inclui a definição de etapas que permitem monitorizar o estado do projeto e, consequentemente, reavaliar datas de entregas ou até mesmo que documentos serão entregues em cada uma das etapas. Ajuda também na identificação de pontos de estrangulamento no caso de projetos mais complexos em que existem várias fases interdependentes. Para além disso, uma vez que através de um bom plano é possível perceber bem em que fase o projeto se encontra, a verificação da existência de desvios consideráveis em relação ao planeamento original pode fornecer informações importantes para o planeamento de projetos no futuro com uma maior exatidão.

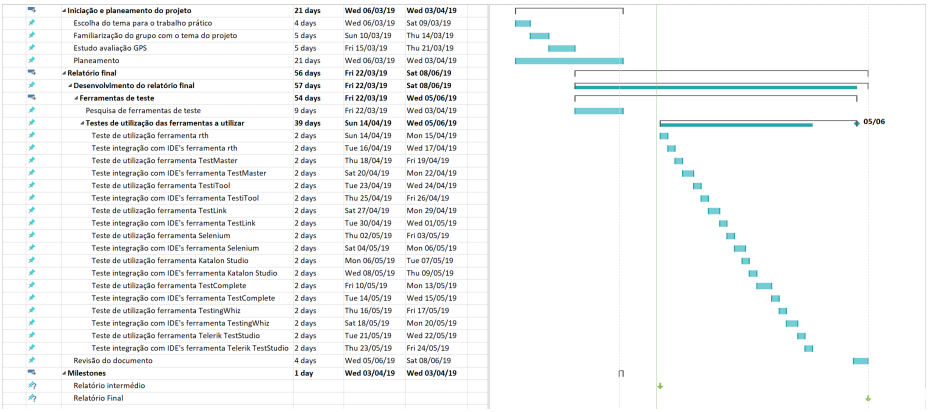
Tendo em conta, agora, as partes interessadas do projeto, um bom plano de projeto permite também que se reflitam da melhor forma possível todas as exigências de cada um deles também considerando a sua relevância no projeto.

Finalmente, a implementação de um plano de contingência que tenha em conta o calendário e o orçamento atribuído ao projeto é essencial uma vez que devem existir soluções previamente pensadas para possíveis problemas que possam surgir no decorrer do projeto. A falta deste, faz com que o aparecimento de um problema provoque o não cumprimento de etapas previamente definidas ou até levar a que o custo do projeto aumente, o que não é, de todo, desejável.

Concluindo, um plano de desenvolvimento deve:

- Definir as tarefas a realizar, como e quando;
- Definir quem é responsável por cada tarefa;
- Um mecanismo de monitorização do progresso do projeto;
- Espelhar os interesses dos *stakeholders*;
- Um plano de contingência;
- Definição dos meios de comunicação.

Apresenta-se, então, o plano desenvolvido para a primeira fase deste projeto, que se baseou na pesquisa e levantamento de ferramentas, com a futura previsão do tempo necessário para a análise de cada uma.

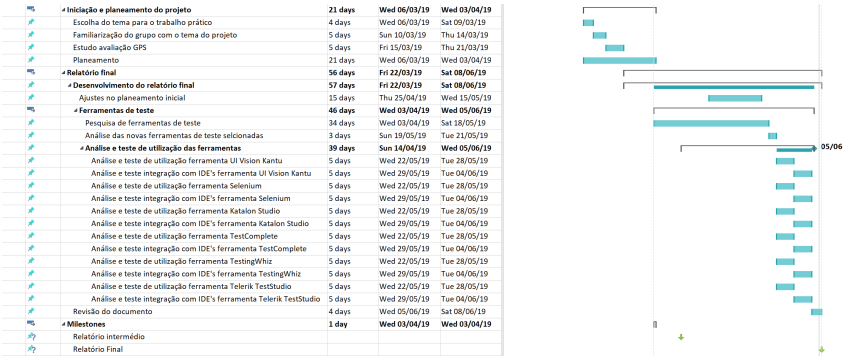


Iniciação e planeamento do projeto	**21 days**	**Wed 06/03/19**	**Wed 03/04/19**
Escolha do tema para o trabalho prático	4 days	Wed 06/03/19	Sat 09/03/19
Familiarização do grupo com o tema do projeto	5 days	Sun 10/03/19	Thu 14/03/19
Estudo avaliação GPS	5 days	Fri 15/03/19	Thu 21/03/19
Planeamento	21 days	Wed 06/03/19	Wed 03/04/19
Relatório final	**56 days**	**Fri 22/03/19**	**Sat 08/06/19**
Desenvolvimento do relatório final	**57 days**	**Fri 22/03/19**	**Sat 08/06/19**
Ferramentas de teste	**54 days**	**Fri 22/03/19**	**Wed 05/06/19**
Pesquisa de ferramentas de teste	9 days	Fri 22/03/19	Wed 03/04/19
Testes de utilização das ferramentas a utilizar	**39 days**	**Sun 14/04/19**	**Wed 05/06/19**
Teste de utilização ferramenta rth	2 days	Sun 14/04/19	Mon 15/04/19
Teste integração com IDE's ferramenta rth	2 days	Tue 16/04/19	Wed 17/04/19
Teste de utilização ferramenta TestMaster	2 days	Thu 18/04/19	Fri 19/04/19
Teste integração com IDE's ferramenta TestMaster	2 days	Sat 20/04/19	Mon 22/04/19
Teste de utilização ferramenta TestiTool	2 days	Tue 23/04/19	Wed 24/04/19
Teste integração com IDE's ferramenta TestiTool	2 days	Thu 25/04/19	Fri 26/04/19
Teste de utilização ferramenta TestLink	2 days	Sat 27/04/19	Mon 29/04/19
Teste integração com IDE's ferramenta TestLink	2 days	Tue 30/04/19	Wed 01/05/19
Teste de utilização ferramenta Selenium	2 days	Thu 02/05/19	Fri 03/05/19
Teste integração com IDE's ferramenta Selenium	2 days	Sat 04/05/19	Mon 06/05/19
Teste de utilização ferramenta Katalon Studio	2 days	Mon 06/05/19	Tue 07/05/19
Teste integração com IDE's ferramenta Katalon Studio	2 days	Wed 08/05/19	Thu 09/05/19
Teste de utilização ferramenta TestComplete	2 days	Fri 10/05/19	Mon 13/05/19
Teste integração com IDE's ferramenta TestComplete	2 days	Tue 14/05/19	Wed 15/05/19
Teste de utilização ferramenta TestingWhiz	2 days	Thu 16/05/19	Fri 17/05/19
Teste integração com IDE's ferramenta TestingWhiz	2 days	Sat 18/05/19	Mon 20/05/19
Teste de utilização ferramenta Telerik TestStudio	2 days	Tue 21/05/19	Wed 22/05/19
Teste integração com IDE's ferramenta Telerik TestStudio	2 days	Thu 23/05/19	Fri 24/05/19
Revisão do documento	4 days	Wed 05/06/19	Sat 08/06/19
Milestones	**1 day**	**Wed 03/04/19**	**Wed 03/04/19**
Relatório intermédio			
Relatório Final			

Figura 4: Plano de desenvolvimento para o projeto proposto apresentado na fase intermédia.

Uma vez que a fase anteriormente entregue apenas contemplava a reunião de ferramentas indicadas para a execução de testes de sistema, inicialmente o grupo pensou em apresentar uma análise a fundo de todas estas. No entanto, tendo em conta o seu número, foram selecionadas apenas seis. Esta decisão foi tomada por forma a que os elementos do grupo pudessem efetuar uma investigação mais a fundo a cada uma das ferramentas, mais concretamente, funcionalidades disponibilizadas, análise das interfaces de cada uma, requisitos de sistema requeridos por estas para a sua execução, modo de integração no ambiente de desenvolvimento, entre outras.

Mais ainda, a ferramenta *Test Link* foi inicialmente uma das ferramentas escolhidas para uma análise mais em detalhe mas, visto que o foco desta era principalmente a gestão dos testes em si e não a integração da execução de testes no ambiente de desenvolvimento, esta foi removida da investigação e no seu lugar surge então o *Ui Vision Kantu*, apresentando funcionalidades muito interessantes para esta área de investigação. Como este caso, também a ferramenta *Katalon Studio*, de inicial interesse, foi alterada pela mesma razão, surgindo a ferramenta UFT. Assim, tendo em conta estes contratempos, juntamente com alguns atrasos, surgiu um novo plano, tal como se pode observar a seguir.



	Iniciação e planeamento do projeto	**21 days**	**Wed 06/03/19**	**Wed 03/04/19**
	Escolha do tema para o trabalho prático	4 days	Wed 06/03/19	Sat 09/03/19
	Familiarização do grupo com o tema do projeto	5 days	Sun 10/03/19	Thu 14/03/19
	Estudo avaliação GPS	5 days	Fri 15/03/19	Thu 21/03/19
	Planeamento	21 days	Wed 06/03/19	Wed 03/04/19
	Relatório final	**56 days**	**Fri 22/03/19**	**Sat 08/06/19**
	Desenvolvimento do relatório final	**57 days**	**Fri 22/03/19**	**Sat 08/06/19**
	Ajustes no planeamento inicial	15 days	Thu 25/04/19	Wed 15/05/19
	Ferramentas de teste	**46 days**	**Wed 03/04/19**	**Wed 05/06/19**
	Pesquisa de ferramentas de teste	34 days	Wed 03/04/19	Sat 18/05/19
	Análise das novas ferramentas de teste selcionadas	3 days	Sun 19/05/19	Tue 21/05/19
	Análise e teste de utilização das ferramentas	**39 days**	**Sun 14/04/19**	**Wed 05/06/19**
	Análise e teste de utilização ferramenta UI Vision Kantu	5 days	Wed 22/05/19	Tue 28/05/19
	Análise e teste integração com IDE's ferramenta UI Vision Kantu	5 days	Wed 29/05/19	Tue 04/06/19
	Análise e teste de utilização ferramenta Selenium	5 days	Wed 22/05/19	Tue 28/05/19
	Análise e teste integração com IDE's ferramenta Selenium	5 days	Wed 29/05/19	Tue 04/06/19
	Análise e teste de utilização ferramenta Katalon Studio	5 days	Wed 22/05/19	Tue 28/05/19
	Análise e teste integração com IDE's ferramenta Katalon Studio	5 days	Wed 29/05/19	Tue 04/06/19
	Análise e teste de utilização ferramenta TestComplete	5 days	Wed 22/05/19	Tue 28/05/19
	Análise e teste integração com IDE's ferramenta TestComplete	5 days	Wed 29/05/19	Tue 04/06/19
	Análise e teste de utilização ferramenta TestingWhiz	5 days	Wed 22/05/19	Tue 28/05/19
	Análise e teste integração com IDE's ferramenta TestingWhiz	5 days	Wed 29/05/19	Tue 04/06/19
	Análise e teste de utilização ferramenta Telerik TestStudio	5 days	Wed 22/05/19	Tue 28/05/19
	Análise e teste integração com IDE's ferramenta Telerik TestStudio	5 days	Wed 29/05/19	Tue 04/06/19
	Revisão do documento	4 days	Wed 05/06/19	Sat 08/06/19
	Milestones	**1 day**	**Wed 03/04/19**	**Wed 03/04/19**
	Relatório intermédio			
	Relatório Final			

Figura 5: Plano final de desenvolvimento do projeto realizado.

5 Conclusões e Trabalho Futuro

Como foi introduzido no início deste estudo, o processo de desenvolvimento envolve um conjunto alargado de etapas onde as oportunidades de injeção de falhas são muito grandes, falhas na sua grande maioria humanas. Deste modo, é importante ter ferramentas que auxiliem na aplicação de técnicas de desenvolvimento *Agile*.

Noutra perspetiva, uma só ferramenta não é capaz de efetuar testes de sistema aplicáveis a todos os *softwares* desenvolvidos e a serem desenvolvidos futuramente. Cada projeto tem uma determinada dimensão, abordagem, estrutura e implementação. É necessário estudar as ferramentas que o mercado atual oferece e adequar as suas caraterísticas e funcionalidades ao *software* a ser desenvolvido.

Deste modo, o estudo efetuado sobre as principais ferramentas existentes permitiu estabelecer os pontos fortes e fracos de cada ferramenta e em que contexto estas devem ser aplicadas. Ainda mais à frente, a integração destas ferramentas com ambientes de desenvolvimento integrado (IDE) é crucial para a aplicação de técnicas *Agile*, na medida em que o contínuo desenvolvimento do projeto é acompanhado por um conjunto de testes que permitem a deteção de erros em fases precoces do desenvolvimento.

Ferramentas como o *Selenium* dominam uma grande fatia do mercado por ser bastante abrangente e capaz de atender a pequenos e grandes projetos.No entanto, outras ferramentas como o *Unified Funcional Testing*, devido à sua elevada curva de aprendizagem, pode ser considerada mais adequada a projetos densos e que permitam ao utilizador tem grande controlo sobre o conjunto de testes a efetuar. Há ainda ferramentas como o *TestComplete* que integram tecnologias emergentes como Inteligência Artificial na construção de testes que auxiliem a deteção de falhas.

O estudo das ferramentas de teste de sistema a utilizar deve ter em conta as caraterísticas do *software* desenvolvido, mas não só. É estritamente necessário observar a evolução e os desafios que o contexto computacional nos coloca e adaptar os testes efetuados consoante essa observação.

Referências

1. Logical Minds. *Ferramentas de Teste de Software: Quais as melhores?*, http://logicalminds.com.br/ferramentas-de-teste-de-software-quais-as-melhores/, 12/01/2018.
2. Wanessa Silva, Angélica Calazans. *Ferramentas free para teste de software: um estudo comparativo*, https://www.publicacoesacademicas.uniceub.br/gti/article/view/1956/1789, Centro Universitário de Brasília, 25/06/2012.
3. Fabio Santos, Gabriel Daltro, Jefferson Couto. *Ferramentas de Teste de Software*, http://pesquompile.wikidot.com/ferramentas-teste-sw, Universidade Católica do Salvador, 17/10/2011.
4. Michelle. *The Importance of Project Planning For Successful Outcomes*, https://www.parallelprojecttraining.com/blog/importance-project-planning-project-success/, Parallel Project Training, 28/02/2018.
5. TestingWhiz, https://www.testing-whiz.com/
6. TestComplete, https://smartbear.com/product/testcomplete/overview/
7. UI Vision, https://ui.vision/
8. UFT, https://www.microfocus.com/pt-br/products/unified-functional-automated-testing/overview
9. Telerik Test Studio, https://www.telerik.com/teststudio
10. Selenium, https://www.seleniumhq.org/