Mestrado em Engenharia Informática
Dissertação/Estágio
Relatório Final

Quality Management Platform

Tiago Filipe Guerreiro Sousa Santiago

tfgss@student.dei.uc.pt

Orientadores:

Miguel Antunes – RedLight Software

Prof. Alberto Cardoso - DEI

Data: 9 de Setembro de 2013

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Resumo

Dado o aumento considerável do número de empresas a actuar na área de Engenharia de Software, estas têm encontrado na garantia de qualidade uma forma de distinção das demais, seguindo os princípios e modelos de maturidade sólidos e com resultados comprovados. O certificado de qualidade do modelo CMMI institui e demonstra a aptidão das organizações para desenvolver software de forma sistemática, controlada e de qualidade.

Contudo a competência técnica presente no desenvolvimento de software é apenas uma das componentes necessárias ao sucesso de uma empresa. Uma empresa que pretenda como um todo assegurar conformidade dos seus produtos e serviços, satisfazer as necessidades e expectativas dos seus clientes e implementar acções de melhoria contínua, terá de fazer uma gestão de qualidade implementada por uma Sistema de Gestão de Qualidade. O certificado de qualidade ISO 9001:2008 descreve um conjunto de requisitos que visam implementar e demonstrar a capacidade eficiente, consistente e transparente do Sistema de Gestão de Qualidade da empresa em atingir os seus objectivos. A empresa RedLight Software pretende certificar-se com ISO 9001:2008 e CMMI para que sejam reconhecidos e comprovados os seus elevados padrões de qualidade de forma transparente.

O presente estágio enquadra-se precisamente neste objectivo estratégico da empresa de acolhimento, centrando-se na análise das necessidades para o cumprimento dos requisitos impostos pelas duas certificações, de onde resultou a completa especificação e implementação de dois processos de Engenharia CMMI de nível 3: Os processos de "V&V" (Verificação e Validação). Os resultados esperados deste trabalho traduzem-se sobretudo na capacidade da empresa aumentar os padrões de qualidade dos produtos desenvolvidos, e simultaneamente reduzir o *overhead* e *rework* das actividades operacionais associadas ao processo de desenvolvimento de software.

Palavras-Chave

"ISO", "CMMI", "QMS", "Processos", Qualidade, "Verificação", "Validação", "Engenharia de Software".

Tabela de Acrónimos

Acrónimo	Significado
BRS	*Business Requirements Specification*
CMMI	*Capability Maturity Model Integration*
COTS	*Commercial off-the-shelf*
ISO	*International Organization for Standardization*
KPI	*Key Performance Indicator*
PM	*Project Manager*
QM	*Quality Manual*
QMS	*Quality Management System*
RLS	*RedLight Software*
SRS	*Software Requirements Specification*
YRS	*System Requirements Specification*

"Intellectuals solve problems; geniuses prevent them."

Albert Einstein

Dedicado a todos aqueles que me apoiaram neste longo percurso….

Lista de anexos confidenciais

Nota: A leitura deste relatório deverá ser feita, sempre que possível, em paralelo com respectivos anexos.

A1- Quality Manual
A2- Mission Management
A3-Documents Management

B1- Verification Process
B2- Documents Review

C1- Validation Process
C2- Test Planning
C3- Test Design and Implementation
C4- Unit Testing
C5- Integration Testing
C6- System Testing
C7- Acceptance Testing
C8- Test Review

D1- Checklists
D2- Code Review - Author
D3- Code Review - Reviewers
D4- Design Review
D5- GUI Review

D6- Requirements Elicitation Review
D7- Requirements Review

E1- Templates
E2- Document Review Meeting Minutes
E3- Document Review Report
E4- Requirements Traceability Matrix
E5- Software Testing Report
E6- Test Plan

F1- Document Review Meeting Minutes - 23-05-2013

G1- Work Instructions
G2- Confluence Tasks and JIRA Issues
G3- Documents Formatting

H1- Business Requirements Spec
H2- System Requirements Spec
H3- Contextual Requirements
I1- CHECKLIST ISO 9001:2008

Índice

Capítulo 1
1.1. Entidade orientadora

A Redlight Software (RLS) é uma *startup* dedicada à Engenharia de Software e Sistemas. A sua faceta diferenciadora, bem presente no seu logótipo, está na ênfase que é dado ao factor humano. Estando sua génese tão intrinsecamente ligada a este factor, a RLS foca a sua operação em áreas de grande impacto humano tal com *Health and Safety*, fortalecendo ainda mais este empenho através da importância dada à usabilidade dos seus sistemas.

A Redlight Software conta com colaboradores vindos das diversas áreas de Engenharia de Software, desde algoritmia pura, até ao *design* de interfaces, ergonomia e processos de engenharia. Conta, entre os seus fundadores, com investigadores com fortes ligações à Universidade de Carnegie Mellon (USA), berço da arquitectura de *software* e do desenvolvimento de *software* como processo de engenharia (CMMI), bem como profissionais com longa experiência na indústria e em design de interacção.

O *core* da actividade da Redlight Software é o desenvolvimento de soluções à medida, para clientes exigentes com a qualidade dos seus sistemas informáticos e na integração com as actividades das suas organizações. Desde sistemas de informação completos a aplicações móveis, o foco está na usabilidade dos interfaces e na fácil integração dos sistemas informáticos com os seus utilizadores finais.

1.2. Contexto do estágio

O estágio apresentado enquadra-se nos objectivos estratégicos da empresa RedLight Software (RLS) em definir o seu QMS (Quality Management System), e desenhar e implementar/integrar o SI (Sistema de Informação) que irá permitir a gestão e controlo deste. É também objectivo cumprir os requisitos de certificação de Qualidade ISO 9001:2008 e do modelo CMMI.

A particularidade e ambição patente em alguns dos objectivos de mais alto nível da empresa (como o desenho de mecanismos de melhoria contínua transversais à empresa e com a contribuição dos seus colaboradores de forma não-entrópica, ou a capacidade de controlo de fluxo de processos e recolha automática de registos e métricas) tornam o estágio apresentado num desafio de grande dimensão. Pretende-se assim definir 2 processos CMMI basilares da área de Engenharia de Software, e ser capaz de especificar um sistema de informação de suporte ao QMS da empresa, que seja capaz de garantir um aumento significativo da qualidade e sucesso dos produtos realizados sem introduzir o *overhead* normalmente associado ao nível de formalismo pretendido.

O QMS irá crescer seguindo as normas e modelos de dois importantes certificados de qualidade: ISO9001:2008 [1] que será aplicado a toda a organização em geral, e CMMI que tem como propósito ser aplicado na área de *Software Engineering*. Estes dois certificados serão seguidos e suportados pelo QMS, com os objectivos de acrescentar valor, eficiência e garantir a excelência de qualidade em todos os produtos e serviços desenvolvidos pela empresa.

Na área de *Software Engineering*, o QMS, desenhado de acordo com o modelo de maturidade CMMI, deverá sobretudo definir e normalizar processos de desenvolvimento de *software*. Estas acções são vitais, já que a qualidade do *software* está fortemente dependente da qualidade dos

seus processos de desenvolvimento. O facto de serem definidos padrões nos processos de desenvolvimento de *software* irá possibilitar a recolha de métricas e artefactos de diferentes projectos e actividades, permitindo à organização certificar-se de que os objectivos definidos estão a ser cumpridos, inclusive através de políticas de melhoria contínua (previstas nas normas referidas).

A norma ISO9001:2008 tem como objectivo definir padrões de gestão internacionalmente reconhecidos e garantir que os requisitos definidos pela organização possam ser cumpridos. Os requisitos poderão ser impostos pelo negócio, pelas normas e práticas de qualidade, eficiência ou mesmo requisitos provenientes de clientes. Essa garantia será dada pelo Sistema de Informação que gere o QMS, segundo as práticas definidas nos processos operacionais implementados pelo mesmo, que irá controlar e assegurar, entre outras coisas, gestão documental, verificar que os processos da empresa têm capacidade em atingir os resultados planeados e descrever e assegurar registos adequados para serem usados em auditorias de qualidade internas e externas, e no confronto com conformidades e não conformidades descritas nas normas ou derivadas dos objectivos e necessidades da organização.

1.3. Âmbito, Objectivos e Motivações do Estágio

Actualmente é possível encontrar uma grande variedade de empresas e respectivas soluções de software no mercado. A competição é cada vez maior e a tolerância a defeitos cada vez menor, havendo uma descredibilização imediata e quebra na reputação junto dos clientes sempre que um compromisso não é cumprido. Estas condições elevam as exigências dos clientes e a única resposta possível é a garantia de excelência na qualidade dos produtos e serviços prestados pelas organizações, e também o cumprimento dos requisitos prometidos, quer estes sejam prazos ou orçamentos para projectos, requisitos técnicos e funcionalidades nos produtos e serviços, ou mesmo o apoio ao cliente.

Com vista a responder às necessidades da organização no que diz respeito à sua actividade e à garantia de qualidade dos produtos desenvolvidos e serviços prestados, o estágio proposto integra-se no contexto da especificação do QMS da empresa em conformidade com ISO 9001:2008, e na definição e implementação dos processos Verification e Validation (V&V) do modelo CMMI.

Os objectivos do estágio são:

- Analisar os requisitos das normas de qualidade envolvidas (ISO9001:2008 e CMMI)
- Analisar os requisitos e especificações de implementação de um QMS com conformidade com ISO9001:2008 e CMMI.
- Analisar, identificar e levantar os requisitos de negócio e sistema de um SI.
- Especificar um conjunto de mecanismos e ferramentas que sejam necessárias para monitorizar, controlar e garantir o cumprimento dos requisitos definidos (não só pelos certificados de qualidade ISO9001:2008 e CMMI mas também pelas motivações e necessidades da organização)
- Contribuir para a estruturação do QMS da RLS.
- Definir completamente os processos de *Verification* (VER) e *Validation* (VAL) de CMMI nível 3 para a RLS.

Com o trabalho desenvolvido espera-se que a RLS obtenha a capacidade de competir no mercado com altos padrões de qualidade, produtividade e soluções à medida das exigências dos clientes e das diferentes áreas de negócio. Os certificados de qualidade irão comprovar

estas mesmas competências, e dar possibilidade de estabelecer novas parcerias de negócio com clientes que exigem padrões de qualidade comprovados e certificados.

Desta forma antevê-se que o âmbito do estágio esteja enquadrado perfeitamente com as exigências do mercado actual e futuro, sendo que o resultado a longo prazo deste estágio será preparar a empresa RedLight Software para as exigências do mercado, colocando-a na rota de uma melhoria continua em busca da excelência na qualidade de todos os seus produtos e serviços prestados.

A grande motivação para o estágio passou pela oportunidade de adquirir um conjunto de conhecimentos e competências da área de conhecimento de Engenharia de Software, onde ainda hoje escassa a qualidade, maturidade e falta de processos operacionais, nomeadamente de desenvolvimento de software, na grande maioria das empresas. Espera-se que os conhecimentos adquiridos ao longo do estágio permitam a capacidade de possuir uma visão ampla, não só da componente de desenvolvimento de software, mas de todo o processo de negócio e Engenharia que suporta o desenvolvimento de software.

1.4. Planeamento do estágio

O estágio está dividido em 7 fases, sendo que algumas delas irão decorrer em paralelo, como indica o diagrama abaixo:

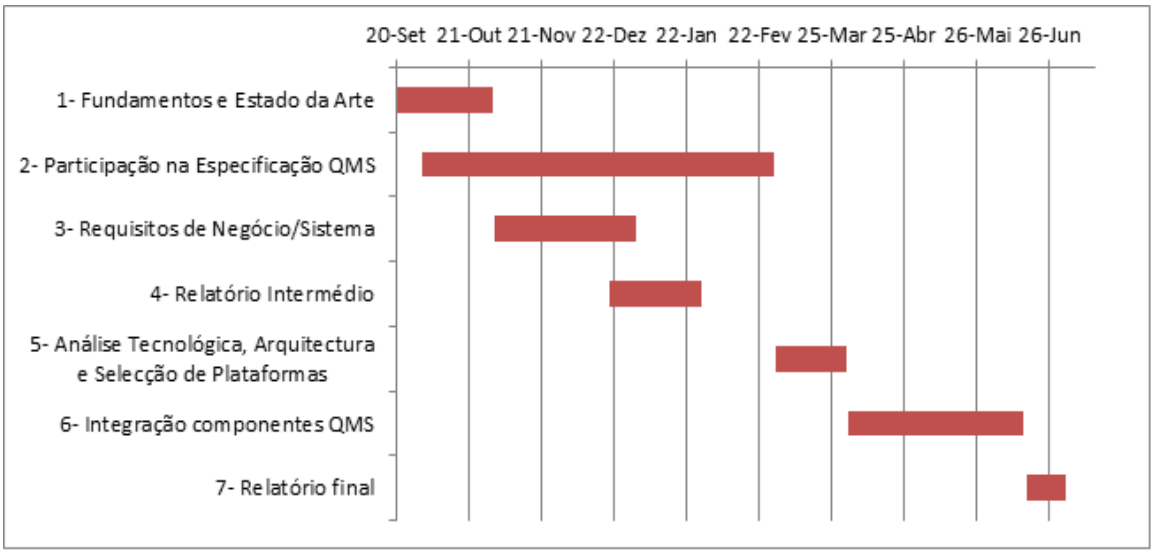


Figura 1 - Diagrama de Gantt para o plano de estágio

Irei seguidamente descrever estas as fases muito brevemente, apenas para ser possível enquadrar o plano. Nos próximos capítulos do relatório, as actividades desenvolvidas durante estas fases serão abordadas com mais detalhe.

Fase 1- Fundamentos e Estado da Arte

A primeira fase do estágio consistiu no estudo de fundamentos e do estado da arte. Nesta fase pretendia adquirir os conhecimentos necessários sobre:

- ISO 9001:2008.
- CMMI.
- QMS.

Fase 2- Participação na especificação do QMS da RedLight

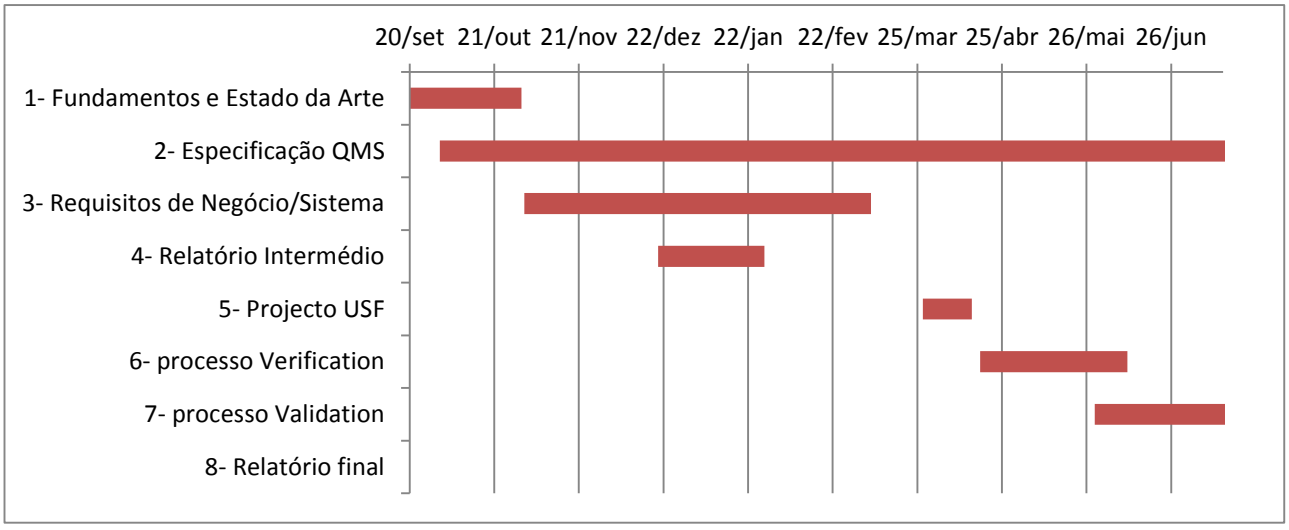
Nesta fase participei activamente na especificação do QMS da RLS, que envolve actividades de estruturação do QMS, como especificação do Manual de Qualidade (QM), ISO *RoadMap Analysis,* definição do Mapa de Processos da organização, definição da Estrutura Organizacional e definição de Processos Operacionais. Pela altura da elaboração deste relatório final a RLS já definiu alguns dos seus Processos Operacionais, estando as restantes actividades completas ou parcialmente completas (sujeitas ainda a futuras revisões ou alterações).

Fase 3- Requisitos

Esta fase teve como objectivo o levantamento e análise dos requisitos de negócio e sistema do projecto da organização. A correcta análise de requisitos tanto de negócio como do sistema é fulcral para o sucesso de um projecto. Se não houver esta análise, ou a mesma for feita de forma incorrecta ou incompleta, irá certamente implicar a curto/médio prazo alterações no projecto que poderão causar várias semanas de atraso, aumento de custos no desenvolvimento e consequente insatisfação do cliente ou utilizador final. As etapas para esta fase são:

- Requisitos de Negócio
- Requisitos de Sistema

Última versão do plano de estágio



Capítulo 2
Estado da Arte

2.1. Qualidade

"Fácil de reconhecer, difícil de definir"[1]. Qualidade pode ter várias definições, consoante a finalidade do produto ou serviço. Na área da saúde, por exemplo, qualidade passa por produtos ou serviços isentos de erros ou falhas (fiabilidade), por outro lado em produtos ou serviços direccionados ao consumidor geral, qualidade poderá significar uma boa relação entre preço e performance ou funcionalidades oferecidas, por exemplo. De uma forma geral a qualidade pode ser definida como o cumprimento dos requisitos e promessas às quais uma organização se compromete a entregar ao cliente. Por exemplo, se imaginarmos dois modelos de carros: Fiat 500L e Ferrari F12, ambos poderão ter ou não qualidade, o importante aqui é não cair no erro de comparar o Fiat ao Ferrari em termos de qualidade, já que ambos poderão ser ao mesmo tempo, carros de alta qualidade dependendo dos parâmetros que lhes foram definidos. A questão aqui é o que a empresa anuncia face ao que realmente oferece (neste exemplo exemplificamos apenas com recurso a macro parâmetros, não incluindo outros mais detalhados como qualidade no processo de construção e linha de produção). Ou seja, a Fiat coloca no mercado um carro que se afirma como sendo de baixo custo, familiar e com baixos consumos. A Ferrari por sua vez anuncia um carro com capacidade de atingir altas velocidades, desportivo, sem ter em conta os consumos ou o preço. Em ambos os casos as marcas cumprem aquilo a que se propõem. Certamente que existem clientes de ambos os modelos, que vão dizer que consideram carros com qualidade, pois vão de encontro às suas necessidades, mesmo que completamente distintas. Essencialmente para existir qualidade, terão de existir produtos ou serviços que satisfaçam as necessidades dos interessados (*stakeholders*[1]), sendo estes clientes, consumidores finais e até a própria organização que desenvolve o produto.

No desenvolvimento de *software* as premissas mantêm-se, sendo que acresce ainda a particularidade de ser comum as organizações falharem na missão de perceber o que realmente o cliente ou consumidor final quer, ou ainda (mais importante), que realmente precisa. No que diz respeito a características comuns na arquitectura de *software* que podem definir e acrescentar valor em termos de qualidade de um produto temos [2,3]:

- *Reliability* – Mede o risco de potenciais falhas no sistema. Em termos de qualidade espera-se um sistema com baixas taxas de indisponibilidade, sem interrupções e inexistência de erros que possam afectar directamente o utilizador final.
- *Efficiency* – Mede a *performance* e escalabilidade de um sistema. Factores como código fonte optimizados e arquitecturas de *software* apropriadas contribuem para uma boa eficiência do sistema.
- *Security* – Mede potenciais vulnerabilidades na segurança do sistema. Código fonte e desenho de inapropriadas arquitecturas são novamente factores importantes.
- *Maintainability* – São avaliadas características como facilidade de realizar manutenção, adaptabilidade e portabilidade. Se houver alterações na equipa de trabalho, nas necessidades do mercado ou nos objectivos do negócio, não devem existir barreiras que invalidem ou dificultem alterações ao sistema.

[1] Conjunto de partes interessadas – Colaboradores, Clientes, Utilizadores – nas práticas, produtos ou serviços da empresa.

- *Size* – Não tendo influência directa na qualidade, em conjunto com outras das características acima descritas poderá dar a possibilidade de obter várias métricas importantes para avaliar a produtividade das equipas.

As características acima descritas pertencem aos *standards* do *CISQ (Consortium for IT Software Quality)*. No entanto é importante referir ainda outras duas características importantes nas exigências do mercado actual e sobretudo, porque se enquadram na Missão da RLS:

- *Testability* – Capacidade de testar artefactos de um sistema. Características como consistência, completude, inequivocidade e mensurabilidade são essenciais para que os artefactos possam ser sujeitos a testes de qualidade.
- *Usability* – Usabilidade é muito mais que possuir boas interfaces e iterações entre utilizador e sistema intuitivas. Usabilidade significa foco nas necessidades do utilizador final, e um estudo prévio dessas mesmas necessidades. Esse foco deverá existir desde das primeiras fases de um projecto e deverá incluir análises contextuais, protótipos e simulações de iterações com o sistema. Existem inclusive algumas metodologias que aconselham a manter por perto das equipas de trabalho potenciais utilizadores do sistema por forma a poderem dar feedback rápido sobre o sistema em desenvolvimento.

No entanto reconhecer características de qualidade não é suficiente. É necessário também reconhecer a importância da qualidade e atribuir parte do esforço da organização para essa mesma importância. Contudo este esforço tem de começar pela gestão de topo da organização, e pela definição de uma estrutura adequada à transformação da organização que vise uma filosofia de ver defeitos e erros como oportunidades de melhoria continua e que envolva todos os colaboradores no respectivo processo de transformação da organização, caso contrário de nada irá valer o esforço [4]. Esta preocupação das organizações com a qualidade está bem patente no crescimento do número de empresas com certificados de qualidade. Iremos seguidamente analisar esse crescimento.

A importância da qualidade na área da Engenharia Informática nos tempos actuais cresceu em resposta à grande crise que os projectos de engenharia informática atravessaram, e ainda atravessam em alguns Países e organizações, nas décadas de 80s e 90s. Estimava-se que por essa altura os custos de manutenção de *software* eram o dobro dos custos do seu próprio desenvolvimento, e que nos anos 90 esses custos teriam aumentado em cerca de 30% [5]. Em meados dos anos 90, estimava-se que 50% dos projectos ficavam operacionais, mas não eram considerados como bem-sucedidos, ou porque ultrapassavam os prazos em cerca de 50% do estimado ou porque uma esmagadora maioria dos grandes produtos acabavam por nem sequer serem utilizados ou não correspondiam as necessidades dos utilizadores [5]. As causas associadas a esta crise devem-se sobretudo à inexistência de métricas (e consequente inexistência de análise), deficiente planeamento de estimativas e custos do projecto, fraca relação e interacção entre a organização e o cliente ou potenciais utilizadores do produto e processos de desenvolvimento de *software adhoc*.

Nos tempos actuais é crucial eliminar custos associados ao mau planeamento e gestão ou devido a más práticas durante os processos de desenvolvimento de *software*, caso contrário as hipóteses de ter sucesso no mercado são praticamente nulas. O mercado do *software* está, assustadoramente, cada vez mais competitivo [18,19]. Na figura abaixo (figura 2) é possível verificar um crescimento na Índia, que anuncia taxas de crescimento de algumas organizações na área de desenvolvimento de *software* acima dos 100% num período de 3 anos [6]!

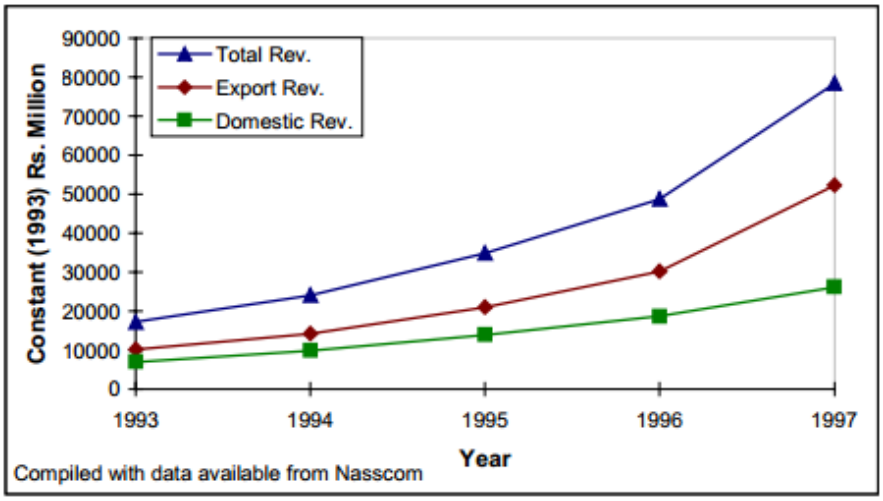


Figura 2 – Indian *Software Industry Revenues (1€ = 71Rs) [6]*

Hoje em dia é portanto fácil perceber a necessidade das organizações se diferenciarem cada vez mais pela excelência na qualidade, e a qualidade significa não só o produto final ir de encontro às necessidades e expectativas dos clientes, como também qualidade na gestão e planeamento de toda a organização, aliás estes dois factores caminham lado a lado. Só assim será possível competir com mercados massivos como a Índia ou a China, onde os custos operacionais são esmagadoramente inferiores (ver figura 3) e onde há uma grande abundancia de recursos humanos especializados. Aliás, iremos ver nas próximas secções que a China, por exemplo, é o país com maior número de empresas com certificados de qualidade…

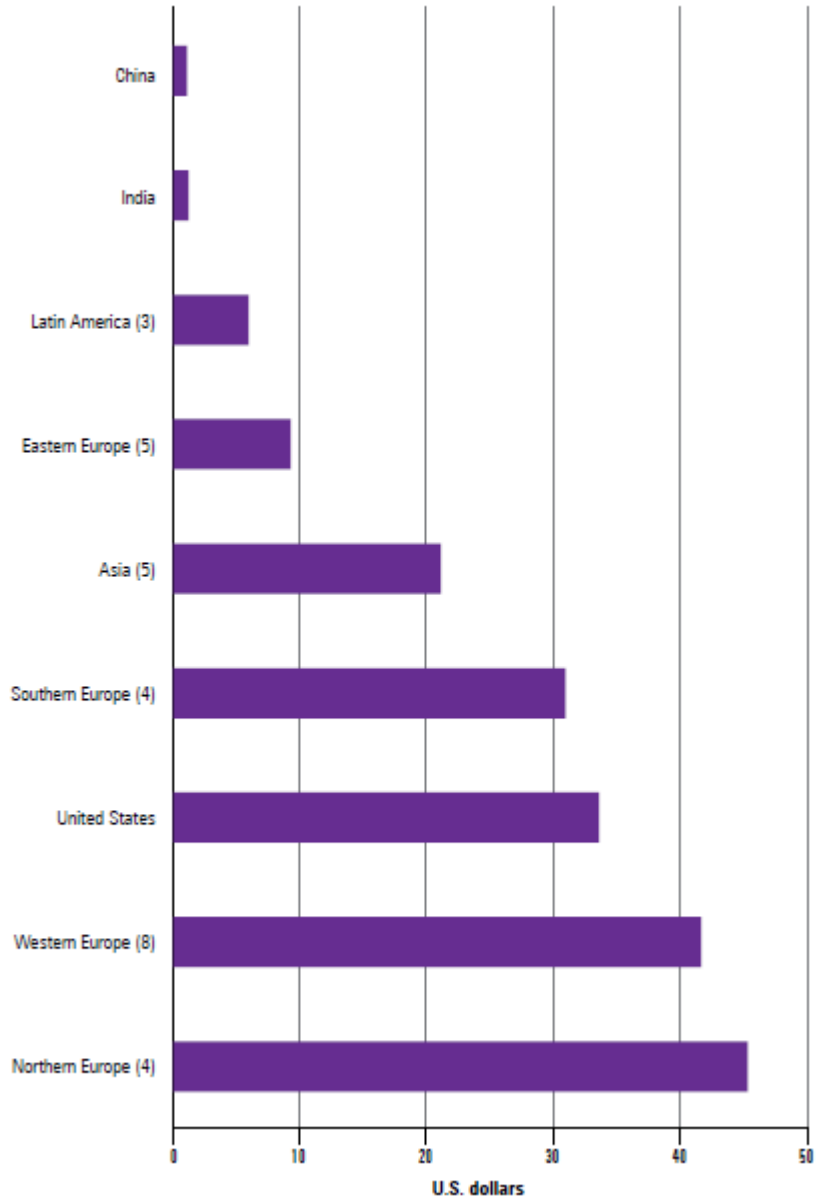


Figura 3 – Custos médios de mão-de-obra *por hora (1 US$ = 0.75€) em 2009 [11]*

2.1.1. ISO 9001:2008

A ISO (International Organization for Standardization) tem o objectivo de conduzir as organizações a normalizarem os seus processos, de uma forma geral, para que as parcerias de negócio internacionais de produtos e serviços sejam mais fáceis, eliminando barreiras culturais, operacionais ou técnicas. Um certificado de qualidade ISO pode ser visto como um cartão-de-visita de uma organização, já que facilmente a identifica em termos de responsabilidades para com a sua gestão de processos e actividades, e por consequência na importância que a organização aplica ao controlo da qualidade dos seus produtos e serviços. Por sua vez, uma organização que se esforça por controlar a qualidade dos seus produtos e serviços, automaticamente terá de se esforçar em perceber as necessidades e requisitos dos seus clientes, já que são eles o propósito da sua existência, e que muitas das vezes são eles que definem ou pelo menos contribuem para os padrões de qualidade dos produtos e serviços da organização.

Por isso mesmo a norma ISO reserva também uma secção muito direccionada à necessidade do foco no cliente por parte das organizações.

A série 9000 da norma ISO tem como propósito ajudar as organizações a desenvolverem os seus Sistemas de Gestão de Qualidade (SGQ) de forma a terem capacidade para atingir os seus objectivos, não sendo limitadas apenas a entregarem produtos ou serviços em conformidade com os requisitos dos seus clientes. Uma organização deverá ter a preocupação e objectivo de deixar não só os seus clientes satisfeitos, mas sim todos os seus *stakeholders*, criando assim uma constante necessidade de melhoria continua na organização.

	ISO 9000	ISO 9004	ISO 19011
Propósito	Facilitar a interpretação dos termos, conceitos e linguagem usada na família das normas	Assistir as organizações na missão de terem um sucesso sustentado em ambientes complexos, exigentes e com alterações constantes.	Assistir as organizações em atingirem uma melhor consistência e eficiência durante auditorias.
Objectivo	Ser usada em conjunto com a ISO 9001 e ISO 9004.	É uma norma descritiva para ser usada apenas como guia. Não deverá ser usada como certificado ou em termos contratuais.	Deve ser usada em auditorias internas e externas de sistemas de gestão.
Scope	Define os princípios, termos e conceitos fundamentais usados na família da ISO 9000	Descreve como as organizações podem atingir um sucesso sustentado aplicando os princípios de gestão de qualidade	Indica os termos de auditorias, como proceder na gestão e planeamento de auditorias.
Aplicação	Aplica-se a todos os termos usados na família das normas ISO 9000	Aplica-se em qualquer organização independentemente do seu tamanho e área de negócio.	Aplica-se a qualquer organização que necessite de planear e gerir auditorias internas ou externas de qualidade.

A norma ISO 9001:2008 (:2008 significa o ano da última revisão) é um *standard* internacionalmente reconhecido, direccionado para os Sistemas de Gestão de Qualidade (QMS) das organizações. A norma especifica um conjunto de requisitos para o QMS com o objectivo de melhorar a sua eficiência, promover a melhoria continua e oferecer maior capacidade em atingir os objectivos planeados e em satisfazer as necessidades dos seus *stakeholders*. No entanto a norma não define como desempenhar as actividades ou quais as características ideais de um produto ou serviço, como tal a norma pode, e deve, ser aplicada a qualquer tipo de organização independente do seu tamanho ou área de negócio. Por ser um

certificado de qualidade, uma organização que pretenda ser certificada ISO 9001 terá de se submeter e ser aprovada em auditorias de entidades certificadas competentes.

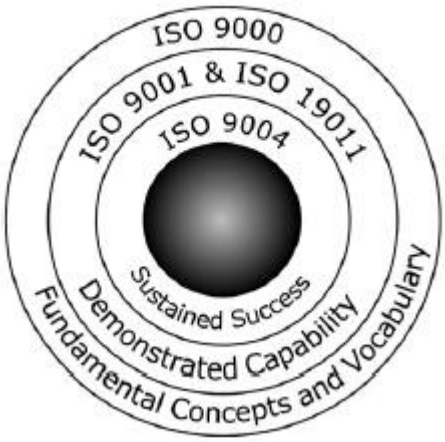


Figura 4 Relação entre a família ISO 9000 [7]

A norma ISO 9001:2008 tem como base 8 princípios de gestão nos quais se acredita estarem intrinsecamente ligados com a dependência da Qualidade: [7]

1. Foco no Cliente

O propósito de uma organização são os seus clientes, portanto a organização terá de compreender as suas actuais e futuras necessidades, corresponder aos seus requisitos e sempre que possível exceder as expectativas dos seus clientes. Uma organização que pretenda aplicar o princípio de *Foco no Cliente* deverá certificar-se que: [7]

- Entende as necessidades e expectativas dos seus clientes.
- Corresponde aos requisitos dos seus clientes, assegurando também que corresponde as necessidades e expectativas de todos os outros *stakeholders*.
- Possui conhecimentos, competências e recursos necessários para satisfazer os seus clientes.
- Recolhe e mede a satisfação dos seus clientes, e actua com apoio nos resultados.
- Entende e gere as relações com os clientes.
- Pode relacionar as suas acções e objectivos directamente com as necessidades e expectativas dos seus clientes.
- É sensível às preferências dos seus clientes, e actua com base na filosofia "o cliente em primeiro lugar".

Para que estes princípios sejam possíveis, a organização deverá possuir na sua base processos específicos para identificar e dar resposta aos requisitos e necessidades dos seus clientes, para que depois possa medir a satisfação e cumprimento dos mesmos.

2. Liderança

A liderança deverá reforçar a Visão e a Missão da organização, e criar um ambiente interno onde os seus colaboradores se sintam envolvidos nos objectivos da organização. Uma organização que siga este princípio de gestão deverá: [7]

- Estabelecer e comunicar uma visão clara do futuro da organização.
- Estabelecer a partilha de valores e éticas transversais a toda a organização.
- Definir objectivos e metas alinhadas com a Missão e Visão da organização.
- Comunicar e implementar estratégias para atingir os respectivos objectivos e metas.
- Utilizar *key performance indicators (KPI)* que encorajem e motivem os colaboradores em atingir os objectivos e as metas.
- Ser pró-activa.
- Entender e responder rapidamente às mudanças de ambiente externas.
- Considerar as necessidades de todos os seus *stakeholders*.
- Criar um ambiente de segurança e confiança, e eliminar o "medo".
- Disponibilizar os recursos necessários e dar a liberdade aos colaboradores de actuar com responsabilidade.
- Promover um ambiente de comunicação aberta e honesta.
- Treinar, educar e ensinar os colaboradores.

A administração da organização deverá ser pró-activa na avaliação do desempenho e disponibilizar os recursos necessários para uma filosofia de melhoria contínua dos seus processos.

3. Envolvimento de Pessoas

As pessoas, de todos os níveis, são a essência da organização e devem ser envolvidas de modo que as suas competências e conhecimentos sejam usados em benefício da organização. Para que este princípio seja cumprido a organização deverá assegurar que os seus colaboradores: [7]

- São pró-activos na busca de oportunidades de melhorias dos seus conhecimentos e competências.
- Aceitam responsabilidades para resolver problemas.
- Partilham livremente os seus conhecimentos e experiências com equipas e grupos de trabalho.
- Têm em mente a criação de valor acrescentado para os clientes da organização.
- São inovadores e criativos no cumprimento dos objectivos da organização.
- São orgulhosos e entusiasmados por fazerem parte da organização.

Estes factores devem ser assegurados pela administração, para que os seus colaboradores sintam que o seu trabalho contribui directamente para os objectivos e qualidade da organização. Devem ser empreendidas acções como *workshops* ou treinos para que os seus colaboradores tenham, e sintam, capacidade em desempenhar as suas funções. A competência dos seus colaboradores também é da responsabilidade da administração.

4. Abordagem por Processos

Os resultados desejados são alcançados de forma mais eficiente quando os recursos e actividades relacionadas são geridos como um processo. Para a organização aplicar uma abordagem orientada a processos deverá: [7]

- Conhecer os objectivos a atingir e identificar os processos que irão dar a capacidade de atingir os resultados necessários.
- Identificar métricas que indiquem quando os objectivos estão a ser atingidos.
- Possuir autonomia e responsabilidade total sobre os resultados.

- Planear e desempenhar apenas as actividades necessárias para atingir os objectivos e resultados planeados.
- Identificar e avaliar riscos para o sucesso e colocar em prática acções preventivas que eliminem, reduzam ou controlem os respectivos riscos.
- Conhecer que recursos, informações e competências são necessários para atingir os objectivos.
- Saber quando um processo está a atingir os seus objectivos.
- Ser pró-activa na busca por melhores formas de atingir os objectivos de um processo, e melhorar a eficiência dos processos.
- Confirmar regularmente se os objectivos e metas continuam a ser necessidades da organização.

Na abordagem por processos é vital a identificação de processos críticos que providenciem valor para os clientes, e todos os outros *stakeholders*. A organização deverá ser responsável não só por garantir o controlo sobre esses processos, como também ser activa na melhoria dos mesmos através da identificação de KPI's, que permitam a avaliação da sua eficácia e eficiência.

5. Abordagem baseada em Sistemas na Gestão

Identificar, compreender e gerir processos interrelacionados, como se de um Sistema se tratasse, irá contribuir para a eficiência e eficácia da organização em atingir os seus objectivos. Uma organização que utilize este princípio de gestão através de uma abordagem por Sistema deverá: [7]

- Ter a capacidade para visualizar as interacções de processos como um Sistema.
- Entender as interacções e interdependências entre os processos e o Sistema.
- Derivar os objectivos do processo a partir dos objectivos do Sistema.

6. Melhoria Contínua

A organização deverá ter como objectivo permanente a melhoria contínua da sua performance. Para a organização aplicar este princípio deverá: [7]

- Estabelecer como objectivo individual para cada colaborador a melhoria contínua de processos, procedimentos e actividades, do Sistema, etc.
- Aplicar os conceitos básicos de melhorias incrementais e melhorias *breakthrough*.
- Utilizar avaliações periódicas nos critérios de excelência para identificar áreas sujeitas a potenciais melhorias.
- Procurar continuamente melhorar a eficiência e eficácia de todos os seus processos.
- Promover acções preventivas.
- Providenciar a cada colaborador a educação e treino, métodos e ferramentas de melhoria contínua.
- Definir métricas e objectivos para monitorizar melhorias.
- Identificar e reconhecer melhorias.

7. Abordagem baseada em Factos para Decisões

Decisões eficazes são baseadas na análise de dados e informações. Para seguir este princípio organização deverá: [7]

- Recolher métricas, dados e informações sobre produtos, processos, actividades, etc.

- Assegurar que os dados e informações são precisos, credíveis e acessíveis.
- Analisar os dados e as informações utilizando métodos apropriados.
- Entender o valor de técnicas de estatísticas apropriadas.
- Tomar decisões e acções com base em resultados de análises lógicas, e utilizando de forma adequada a experiência e intuição nessas decisões e acções.

8. Benefícios Mútuos de Parcerias com Fornecedores

A organização e os seus fornecedores são interdependentes, logo relações onde existam benefícios mútuos irão acrescentar valor a ambos. Para que a organização siga este princípio deverá: [7]

- Identificar e seleccionar fornecedores chave com capacidade para corresponderem aos requisitos sem comprometer na qualidade.
- Criar canais de comunicação abertos e claros.
- Possuir iniciativas de desenvolvimento e melhoramos conjuntos de produtos e processos.
- Em conjunto estabelecer um claro entendimento das necessidades dos clientes.
- Compartilhar informação e planos futuros.
- Reconhecer melhorias e feitos dos seus fornecedores.

Este princípio tem a curiosidade de possuir duas "frentes", já que organização é ao mesmo tempo fornecedor para os seus clientes e um cliente dos seus fornecedores.

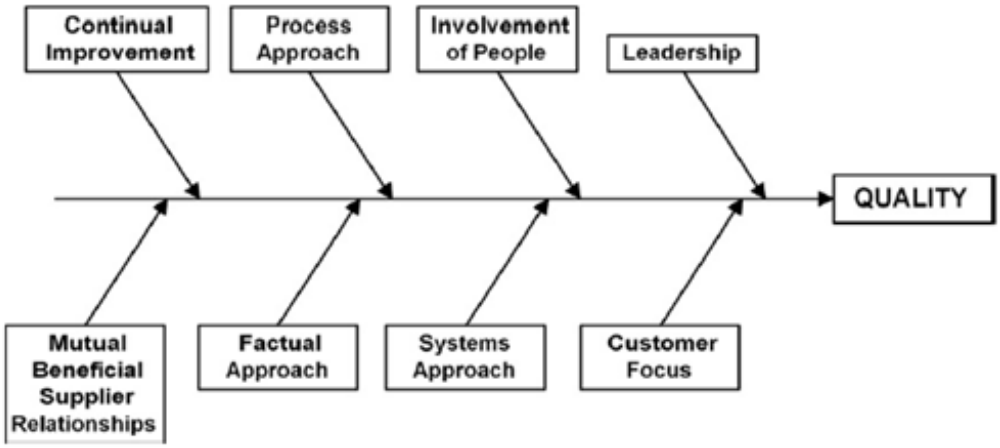


Figura 5 - Diagrama de Ishikawa com os 8 príncipios de gestão de qualidade [7]

Estrutura ISO 9001:2008

A estrutura da norma está dividida em 9 secções e 2 subsecções:

1. A primeira apresenta a Introdução, propósito e âmbito, conteúdos da norma, conceitos e definições, etc.:

    a. **Introdução**
    b. **Objectivo e campo de aplicação**
    c. **Referência normativa**
    d. **Termos e definições**

2. A segunda descreve os requisitos necessários para implementação e monitorização do QMS:

    a. **Sistema de Gestão de Qualidade** – É necessário documentar processos, medir o desempenho e adoptar uma filosofia de melhoria contínua. A organização deve ainda documentar as suas políticas de qualidade e objectivos. Tudo isto deve ser documentado no Manual de Qualidade da organização.
    b. **Responsabilidade da Gestão** – A gestão de topo da organização deve estar envolvida por completo na missão de melhorar a qualidade. Deverá ser responsável por definir as políticas de qualidade e os objectivos. Tem ainda de assegurar que a organização, a todos os níveis, dispõe dos recursos necessários.
    c. **Gestão de Recursos –** A organização deverá ser responsável pelos seus colaboradores (treinos, *workshops*) e possuir instalações e ambiente de trabalho necessárias para atingir os seus objectivos.
    d. **Realização do Produto** – A organização devem possuir processos necessários para identificar os requisitos e necessidades dos seus clientes, planear e desenvolver produtos que correspondam a esses requisitos e necessidades e gerir os seus fornecedores.
    e. **Medição, Análise e Melhoria** – A organização deve monitorizar o cumprimento dos seus objectivos, requisitos e satisfação dos seus clientes. Deverá possuir periodicamente auditorias internas para identificar não-conformidades, mas também ter a capacidade de no decorrer de actividades do dia-a-dia, sempre que necessário, empreender acções correctivas e preventivas que eliminem as causas de não-conformidades. A organização deverá ainda possuir métricas que possibilitem a avaliação de processos e detectar oportunidades de melhorias contínuas no desempenho geral da organização.

Importância e vantagens da ISO 9001

Nas figuras seguintes será demonstrada a importância e crescimento associado de certificados de qualidade ISO 9001 por parte de organizações distribuídas por todo o Mundo.

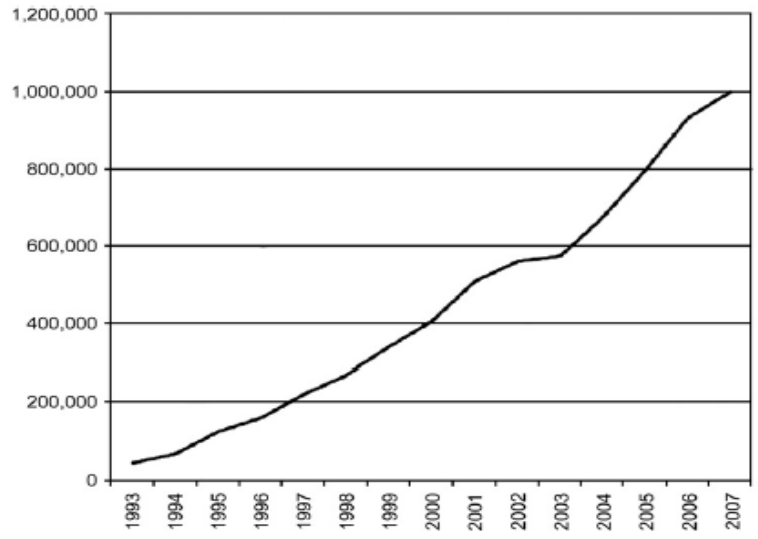


Figura 6 - Crescimento anual ISO 9001 e outros certificados derivados ISO de 1993 a 2007 [7]

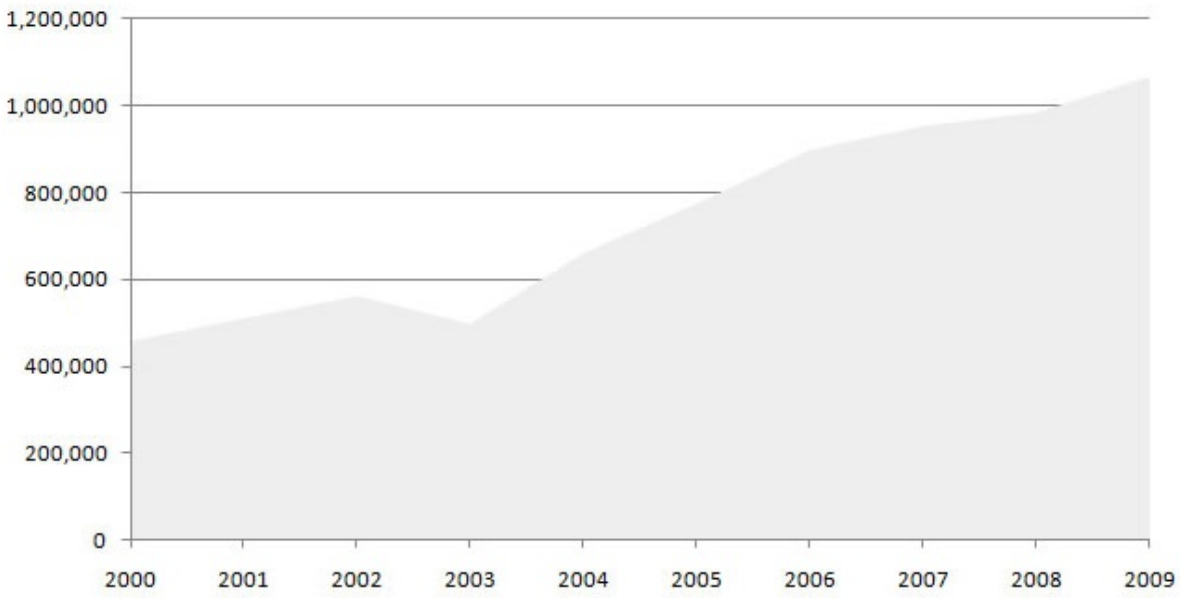


Figura 7 - Crescimento anual ISO 9001 de 2000 a 2009 [8]

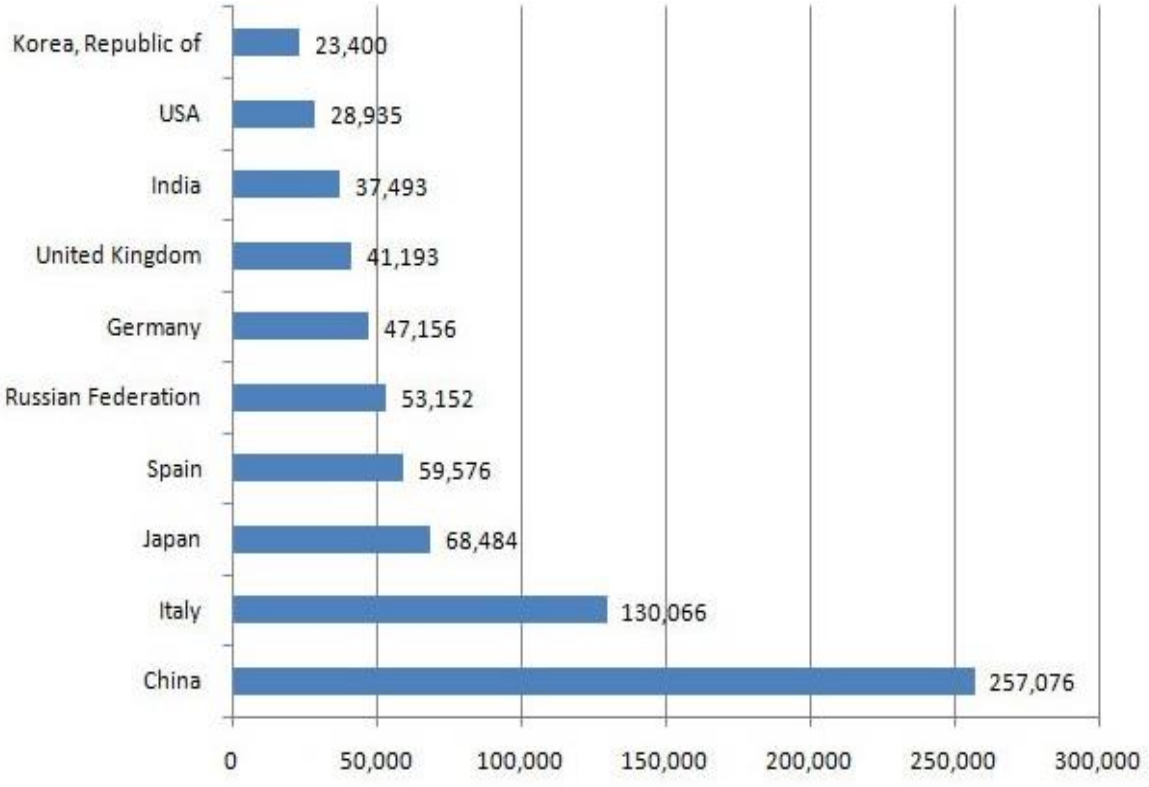


Figura 8 – Top 10 Países com certificações ISO 9001 – Estudo em 2009 [8]

Na figura 6, 7 e 8 é possível confirmar a importância dos certificados de qualidade ISO 9000, pois embora o crescimento tenha tido origem em 1993 ainda é possível verificar nos últimos anos a tendência desse mesmo forte crescimento, o que prova que não se trata de uma moda, mas sim cada vez mais um requisito para com a excelência na qualidade.

Figura 9 – Principais motivos que levaram as organizações a certificar com ISO9001 – Estudo em 2005 pela ABNT [9]

Rank	Benefits	Mean[a]	Std. dev.
1	Increased quality awareness in the firms	4.25	0.66
2	Improved the efficiency of the quality system	4.16	0.76
3	Improved customer service	3.67	0.91
4	Improved the quality of products	3.57	1.13
5	Improved inspection methods and time to produce finished goods	3.53	1.17
6	Reduced customer complaints	3.46	1.02
7	Improved inspection methods and time to receive incoming materials	3.42	1.17
8	Improved the quality of incoming materials	3.36	1.06
9	Reduced defective rate and wastes	3.32	0.89
10	Improved staff motivation	3.30	1.00
11	Improved employee relations	3.14	1.10
12	Improved the speed of good delivery	3.13	1.04
13	Improved productivity	3.06	1.10
14	Improved supplier relations	3.02	1.10
15	Maintained/gained market share	2.98	1.18
16	Improved process design	2.94	1.36
17	Reduced costs	2.87	0.95
18	Improved profitability	2.75	1.10
19	Improved product design	2.54	1.38
20	Increased exports	2.38	1.31

Note: [a]The scale is 5 = crucial effect; 4 = great effect; 3 = some effect; 2 = little effect; 1 = no effect

Figura 10 – Ranking de benefícios atribuídos pelas organizações – [10]

Na figura 9 e 10 é possível verificar os principais motivos e benefícios das organizações em certificarem-se com a norma ISO 9001. Na figura 6 vemos que o aumento de qualidade é um dos principais motivos, ao passo que na figura 7 podemos verificar que os quatro primeiros benefícios são relacionados directamente com qualidade.

2.1.2. CMMI

CMMI é um modelo de maturidade para especificação e melhoria contínua de processos nas áreas de *Software Engineering*, *System Engineering SE, Integrated Product and Process Development IPPD, Supplier Sourcing SS,* que foi desenvolvido na Universidade de Carnegie Mellon *[12]*. Este modelo surgiu em resposta às necessidades do Departamento de Defesa dos Estados Unidos da América que tinha como requisitos produtos e serviços com alta fiabilidade e zero (0) defeitos, já que as consequências de falhas poderiam ser catastróficas. Hoje em dia o modelo CMMI é usado em todo o mundo pelas organizações que pretende ter capacidade para desenvolver produtos e serviços com alta qualidade, e sobretudo que essa alta qualidade seja consistente.

O modelo de maturidade CMMI defende que a qualidade de produtos e serviços está directamente ligada às actividades que fazem parte de todo o processo de desenvolvimento. Como tal, o modelo pretende fazer compreender essa mesma relação entre a qualidade e as actividades desempenhadas para que seja possível melhorar os produtos e serviços da organização, melhorando os seus procedimentos e actividades de desenvolvimento. Trata-se portanto de um modelo focado em processos e colaboradores, no entanto não é objectivo do modelo definir *standards* ou definir a realidade, mas sim representá-la. Isto significa que a organização é livre de escolher as suas abordagens e metodologias, tais como utilizar metodologias de gestão de projecto ágeis ou tipo *waterfall*, ou até poder utilizar folhas de cálculo ou software específico como ferramentas de apoio à gestão. O importante é saber para "onde ir", a forma como fazer esse percurso é da responsabilidade de cada organização.

Maturidade de Processos

São vários os factores dos quais podem depender a maturidade de um processo, mas usualmente podemos dizer que um processo tem um bom grau de maturidade se for bem definido, eficaz, mesurável, analisado e avaliado pela organização, e constantemente sujeito a melhorias contínuas. O modelo CMMI tem como objectivo resolver problemas de maturidade em processos através de um conjunto de boas práticas e modelos que possam melhorar os processos chave da organização.

Na tabela abaixo é possível identificar alguns factores que determinam se uma organização possui maturidade ou não nos seus processos: *[12]*

Factores que determinam maturidade	**Factores que determinam imaturidade**
Procedimentos e actividades desempenhadas de acordo com processos definidos	Processos improvisados durante os projectos
Pró-activa	Reactiva
Projectos concluídos de acordo com o planeamento	Mau planeamento de orçamentos e prazos
Funções e responsabilidades dos colaboradores bem definidos	Qualidade sacrificada em prol do cumprimento de prazos
Processos sujeitos a melhorias contínuas	Inexistência de métricas de qualidade

Representações CMMI

A estrutura do modelo CMMI está organizada por blocos, que são identificados como áreas de processo. Essas áreas de processo não têm como objectivo descrever como deve ser executado um processo eficaz (i.e.: inputs, outputs, roles dos colaboradores, recursos, etc.), mas sim o que executar (práticas) e os seus objectivos. Estas áreas de processo do modelo CMMI podem ser divididas em duas representações: **Contínua** e **Faseada** (*continuous and staged*).

A representação **contínua** trata-se de uma abordagem mais flexível para o processo de melhoria, na medida que permite as organizações escolherem áreas de processo específicas ou processos chaves a melhorar. Esta flexibilidade é especialmente útil quando existe a necessidade de resolver problemas em apenas determinadas áreas da organização, minimizando assim os investimentos e esforços e maximizando benefícios a curto e médio prazo. O desempenho das organizações nesta representação é avaliado sobre a forma de níveis de capacidade de 0 a 5. [12]

A representação **faseada** oferece um *roadmap* detalhado para o processo de melhoria, descrevendo passo a passo a sequência de execução nas áreas de processo, agrupando-as por níveis de maturidade. Este tipo de representação tem a vantagem de garantir uma base sólida de melhorias, preparando assim a organização para o próximo nível, minimizando os riscos, esforços e investimentos. O desempenho das organizações nesta representação é avaliado sobre a forma de níveis de maturidade de 1 a 5. [12]

Na seguinte tabela é possível analisar as vantagens de cada representação: [12]

Representação Contínua	**Representação Faseada**
Capacidade para escolher a ordem de melhorias que melhor se adaptam aos objectivos de negócio da organização.	Permite as organizações seguirem uma sequência de passos pré-definidos e comprovados.
Oferece maior visibilidade da capacidade alcançada dentro de cada área de processo.	Existem históricos de casos de estudo e dados para comprovar os retornos dos investimentos.
Suporta um foco em riscos específicos para áreas de processo individuais	Permite a comparação entre organizações através dos respectivos níveis de maturidades.
Suporta uma melhor comparação de melhoria de processos com a norma ISO 15504, já que a organização das áreas de processo é derivada da ISO 15504.	Introduz uma sequência de melhoramentos, que começam por princípios básicos de gestão que progridem através de sucessíveis níveis, onde cada nível serve de base para o próximo nível.
Permite que as práticas genéricas de níveis de capacidade mais altos sejam aplicadas de forma mais completa e uniforme a todas as áreas de processo.	Sumariza os resultados e capacidade de melhorias de processos em simples níveis de maturidade, numerados de 1 a 5.
	A capacidade e respectivo nível de maturidade da organização são comuns

	serem públicos, e comprovam os seus altos padrões de qualidade.

Inicialmente foi definido na RLS que o *roadmap* seria através da representação faseada, pelos motivos de se tratar de uma *startup* onde existe a necessidade de definir as áreas basilares de processos da organização, e porque a representação faseada exige um esforço progressivo, de acordo com os respectivos níveis, que permitiria ser acompanhado em paralelo pelo crescimento, evolução e maturidade da organização.

No entanto pelas necessidades capturadas durante a análise de requisitos junto dos *stakeholders*, foi tomada a decisão de "atacar" imediatamente as preocupações e necessidades da RLS: Qualidade, *Rework* e *Overhead* associada as actividades de desenvolvimento de software. Deste modo foi tomada a decisão de optar por um *roadmap* baseado na representação contínua, onde foi escolhida a área de processo *Engineering,* nomeadamente a implementação dos processos *Verification* e *Validation* (V&V). Esta decisão justifica-se uma vez que caso fosse optado pela representação faseada, ambos os processos (V&V) apenas poderiam ser implementados no nível 3 de maturidade, ou seja, só quando todos os processos do nível 2 estivessem definidos e complemente implementados (tempo médio de 4 a 11 meses - [16]).

No final os resultados de certificação, quando completas todas as áreas de processo dos respectivos níveis, serão os mesmos dado que ambas as representações convergem para os mesmos objectivos.

Áreas de Conhecimento CMMI

Como foi mencionado anteriormente, embora o modelo CMMI tenha um grande foco na área de desenvolvimento de software, existem ainda outras áreas de conhecimento. As organizações devem portanto fazer uma análise prévia das diferentes áreas, e escolher qual corresponde de melhor forma ao que a organização pretende melhorar. As áreas de conhecimento do modelo CMMI são 4: [13]

- *System Engineering (SE)* – cobre o desenvolvimento total de sistemas, que poderá ou não incluir software. Esta área tem como objectivo transformar as necessidades, expectativas e restrições dos clientes em produtos e soluções, e dar suporte durante todo o ciclo de vida de um produto ou serviço.
- *Software Engineering (SW)* – é direccionado para o desenvolvimento de software. Foca-se na aplicação de abordagens sistemáticas, disciplinadas e quantificáveis para desenvolver, operar e fazer a manutenção de software.
- *Integrated Product and Process Development (IPPD)* – trata-se de uma abordagem sistemática que envolve os *stakeholders* durante o tempo de vida de um produto ou serviço, para aumentar a satisfação dos clientes para com as suas necessidades, expectativas e requisitos. Esta área de conhecimento tem a particularidade de ser integrada com outros processos da organização, como tal se uma organização pretender escolher esta área terá de escolher uma ou mais áreas adicionais, caso elas ainda não existam.
- *Supplier Sourcing (SS)* – Esta área cobre a necessidade dos *project managers* escalarem, quando necessário, esforços de trabalho para *outsourcing*. Esta capacidade poderá ser útil quando os projectos são complexos e existe a necessidade de adicionar funcionalidades ou modificações específicas. No entanto é necessário garantir a análise e monitoramento das actividades desempenhadas por outros parceiros, antes da entrega do produto ou serviço. Esta área exige ser escolhida em conjunto com outros modelos de desenvolvimento de produtos.

A área de conhecimento que responde às necessidades e objectivos da RLS é área de SW, uma vez que é uma empresa focada no desenvolvimento software e serviços, com os objectivos de melhor os seus processos de engenharia de software.

Níveis de Maturidade

Existem 5 níveis de maturidade no modelo CMMI (ver figura 11), onde cada nível é "construído" sobre as bases estabelecidas no anterior, e portanto os níveis formam bases na organização para esta ter a capacidade de atingir o próximo nível. A capacidade de melhoria progride a cada nível, incidindo sobre as informações, métricas e processos da organização dos níveis anteriores.

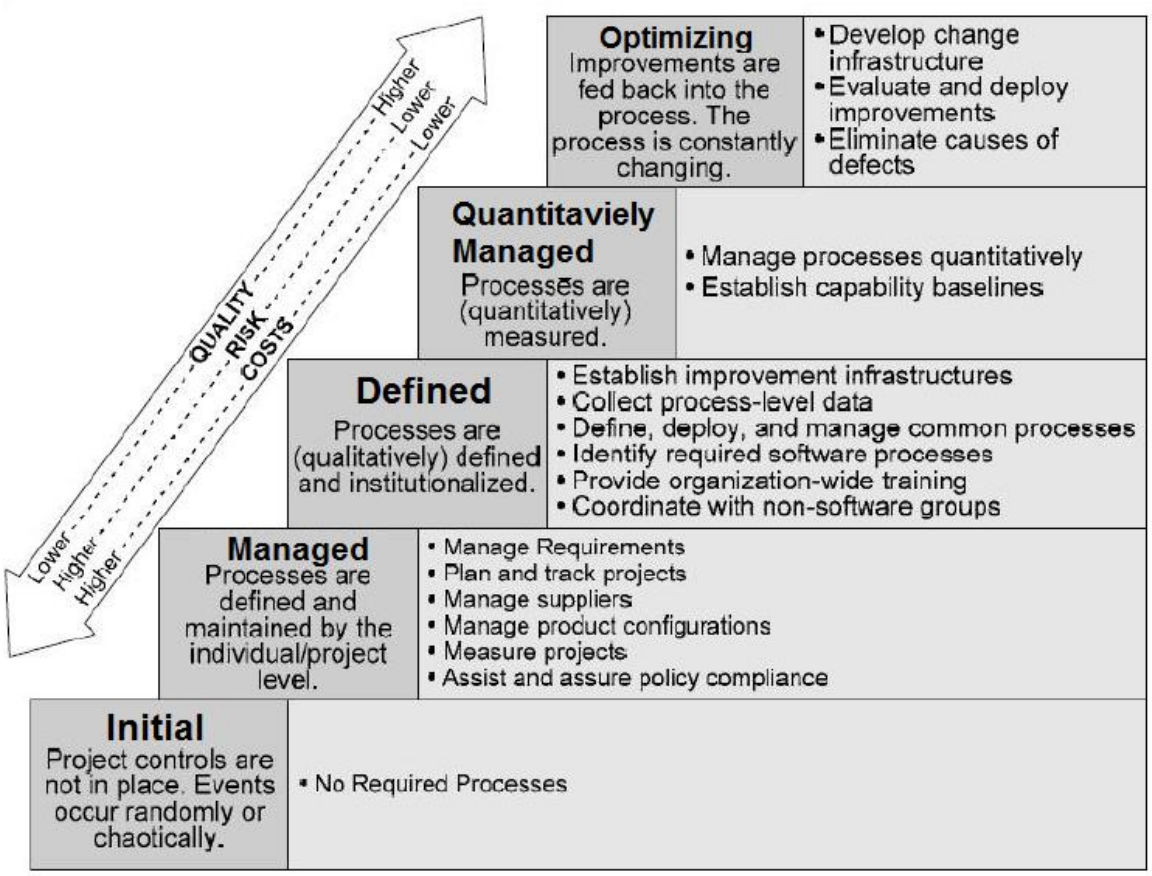


Figura 11 – Características dos 5 Níveis de Maturidade do Modelo CMMI [14]

Nível 1 – *Initial*

No primeiro nível as organizações possuem processos caóticos e usados de forma *ad hoc*. Não existe um ambiente estável e o sucesso da organização depende das competências e actos de "heroísmo" dos seus colaboradores. Este tipo de organizações consegue no entanto produzir produtos e serviços funcionais, mas ultrapassa frequentemente os orçamentos e prazos dos seus projectos. Organizações no nível 1 são portanto caracterizadas por incumprimento de prazos, incumprimento dos processos quando limitadas pelo tempo e por não terem a capacidade de repetir de forma constante os seus sucessos. [13]

Nível 2 – *Managed*

No segundo nível os projectos da organização asseguram a gestão de requisitos e que os respectivos processos são planeados, executados, medidos e controlados. A disciplina de processos presente no nível 2 ajuda as organizações a assegurarem o cumprimento de processos mesmo durante períodos de stress e limitações de tempo, permitindo assim que os projectos sejam executados e geridos de acordo com os planos documentados pela organização. No segundo nível de maturidade todos os requisitos, processos, produtos e serviços são geridos. Os objectivos são estabelecidos entre os *stakeholders* chave da organização e revistos sempre que necessário. Os produtos são também revistos pelos *stakeholders* e controlados pela organização. Este nível de maturidade permite a organização ter a capacidade de satisfazer os requisitos, *standards* e objectivos dos seus produtos e serviços. [13]

Nível 3 – *Defined*

O terceiro nível de maturidade garante que a organização cumpra os objectivos genéricos e específicos nas áreas de processo dos níveis 2 e 3. Organizações neste nível possuem processos bem definidos através de *standards*, procedimentos, ferramentas e métodos. Terá de existir também uma *baseline* definida na organização de *standards*, processos, procedimentos, etc. que seja possível utilizar e adaptar às necessidades e exigências de projecto diferentes ou departamentos da organização. Com isto os processos são executados transversalmente a toda a organização de uma forma consistente, dentro dos padrões estabelecidos pela organização através da sua *baseline*. [13]

Nível 4 – *Quantitaviely Managed*

Neste nível devem ser escolhidos subprocessos que contribuam de forma significativa para a performance geral dos processos. Os subprocessos são controlados usando métricas e métodos estatísticos. São também estabelecidos objectivos quantitativos para ser usados como critérios de gestão e avaliação de processos. Esses objectivos devem ser baseados nas necessidades dos clientes e utilizadores finais, organização e dos próprios processos. A qualidade e a *performance* dos processos são analisadas em termos estatísticos e geridas durante toda a vida de um processo, onde para isso seja necessário recolher métricas de performance e fazer as respectivas análises estatísticas. Sempre que necessário devem ser empreendidas acções que eliminem causas de não-conformidades relacionadas com performance ou objectivos. Uma organização no nível 4 de maturidade tem ainda de possuir um repositório onde seja possível consultar e analisar métricas de qualidade e *performance,* para que no futuro seja possível a organização tomar decisões com base em fatos. Espera-se ainda que no nível 4 a organização tenha a capacidade de prever a *performance* de processos através de técnicas de estatística e métodos qualitativos. [13]

Nível 5 – *Optimizing*

Para atingir o nível 5 de maturidade é necessário cumprir todos os objectivos específicos dos níveis 2, 3, 4 e 5. Neste nível os processos são continuamente melhorados através de melhorias incrementais e inovações. Os objectivos de melhoria de processos são definidos e revistos periodicamente de forma a reflectir mudanças nos objectivos de negócio, e sendo usados na gestão de melhoria de processos. Os efeitos das melhorias são medidos, avaliados e confrontados com os objectivos das melhorias de processos. Toda a *baseline* de *standards* e processos definidos pela organização deverá ser alvo de actividades de melhorias contínuas. A organização deverá garantir que a filosofia de melhorar processos está presente em todos os colaboradores, criando assim um ciclo de melhoria contínua transversal a toda a organização. [13]

Independente do nível de maturidade actual a organização jamais deverá saltar níveis, já que os níveis mais altos de maturidade não só exigem um maior esforço da parte da organização, como também a falta de bases dos níveis anteriores irá aumentar não só o esforço mas os riscos de sucesso. Assim como quando uma organização atinge um certo nível de maturidade, deverá amadurecer os seus processos o suficiente antes de avançar para o próximo nível, garantindo que mesmo em situações de crise e stress a organização cumpre de forma consistente os seus processos.

Níveis de Maturidade vs Áreas de Processo

Uma Área de Processo é um conjunto de práticas relacionadas numa área que quando implementadas colectivamente, satisfazem um conjunto de objectivos importantes para executar melhorias significativas nessa área. As Áreas de Processo podem ser divididas em 4:

- *Process Management*
- *Project Management*
- *Engineering*
- *Support*

E cada área é definida por um conjunto de objectivos e práticas que podem ser divididos em:

- Práticas e Objectivos Genéricos – que fazem parte de todas as áreas de processo
- Práticas e Objectivos Específicos – que são específicos para uma determinada área de processo

Assim sendo, uma área de processo é completa quando os processos da organização cobrem todos os objectivos e práticas genéricas e específicas para essa área de processo.

Process Management

As áreas de processo de *Process Management* contêm as actividades transversais a toda a organização relacionadas com a definição, planeamento, recursos, implementação, monitorização, controlo, apreciação e melhoria.

Project Management

As áreas de processo de *Project Management* cobrem as actividades relativas a planeamento, monitorização e gestão de projecto.

Engineering

As áreas de processo de *Engineering* cobrem o desenvolvimento e manutenção das actividades relacionadas com Engenharia de Software.

Support

As áreas de processo de *Support* envolvem actividades que suportam o desenvolvimento e manutenção dos produtos.

As áreas de processo são assim agrupadas em 4 categorias de PA (*Process Area*) e pelos 5 níveis de maturidade como podemos ver na figura abaixo:

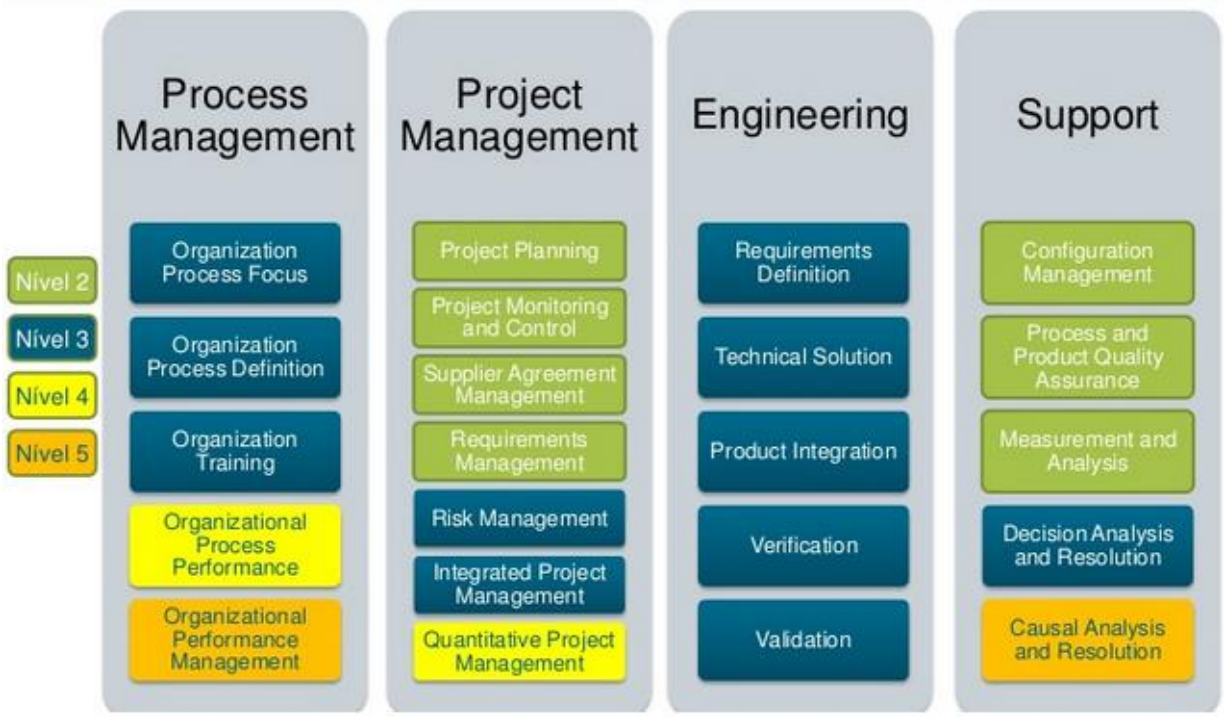


Figura 12 – PA agrupadas pelas 4 categorias e respectivos níveis de maturidade no Modelo CMMI

Importância e vantagens do CMMI

Como foi explicado anteriormente a área de desenvolvimento de software luta constantemente contra problemas nos seus projectos, quer sejam de prazos não cumpridos, custos acima do estimado, erros (bugs) nas versões finais de produtos, *rework*, etc.

Os erros e problemas mais comuns são:

- "Encontrar e resolver um problema de software depois da sua entrega é frequentemente 100 vezes mais caro, que encontrar e resolve-lo durante a fase de requisitos e arquitectura". [14]
- "Os atuais projecto de software gastam mais ou menos 40 a 50% do seu esforço em *rework* evitável". [14]
- "Práticas disciplinadas podem reduzir a "produção" de erros durante o desenvolvimento de software em 75%". [14]
- "Mais de 60% dos erros nos produtos de software são cometidos durante a fase de arquitectura e menos de 40% durante a fase de '*coding*' ". [14]
- "Apenas 60% das funcionalidades de um sistema são realmente utilizadas". [14]

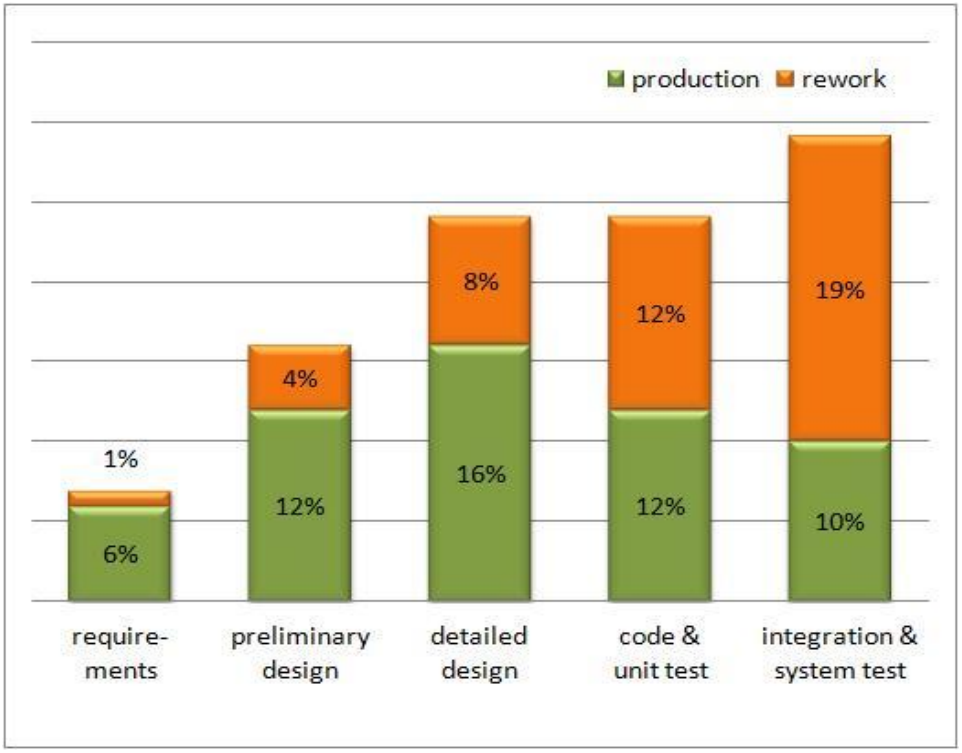


Figura 13 – Esforço no desenvolvimento produtivo e *rework* durante o desenvolvimento de software [15]

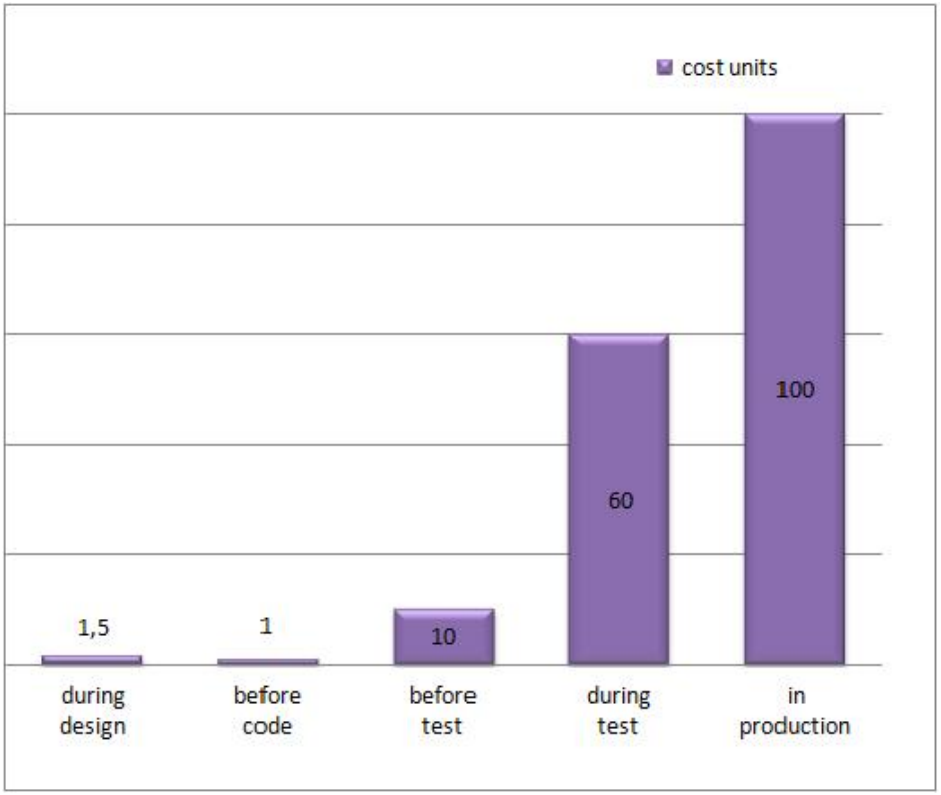


Figura 14 – Custo relativo para resolver um *bug* dependente da fase em que é detectado [15]

Através das citações dos erros e problemas mais comuns no desenvolvimento de software e analisando as figuras anteriores (13 e 14), percebe-se que existe falha das organizações para com a importância da Engenharia de Software. No entanto a situação tem-se vindo a alterar e as organizações cada vez mais na Engenharia de Software, reconhecendo que para competir no mercado é necessário apostar na Qualidade. No caso do modelo CMMI a situação comprova-se:

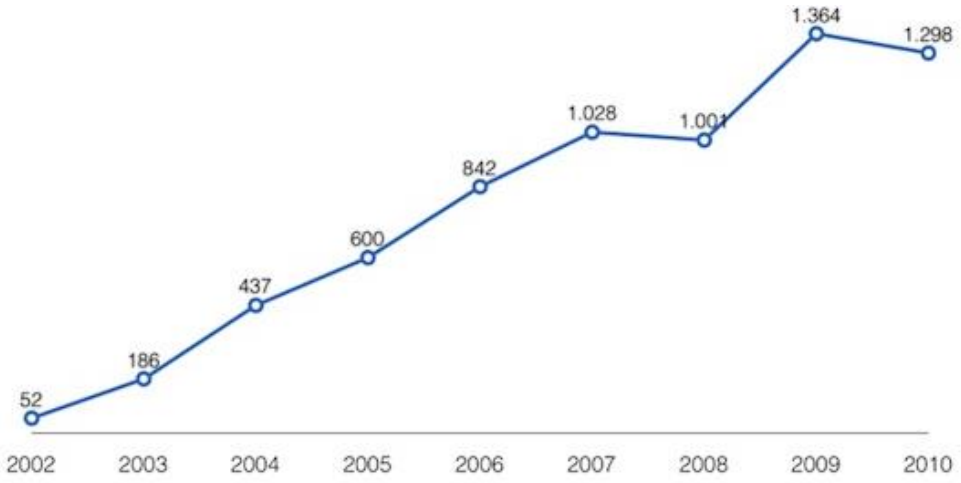


Figura 15 – Crescimento anual de avaliações do modelo CMMI - [16]

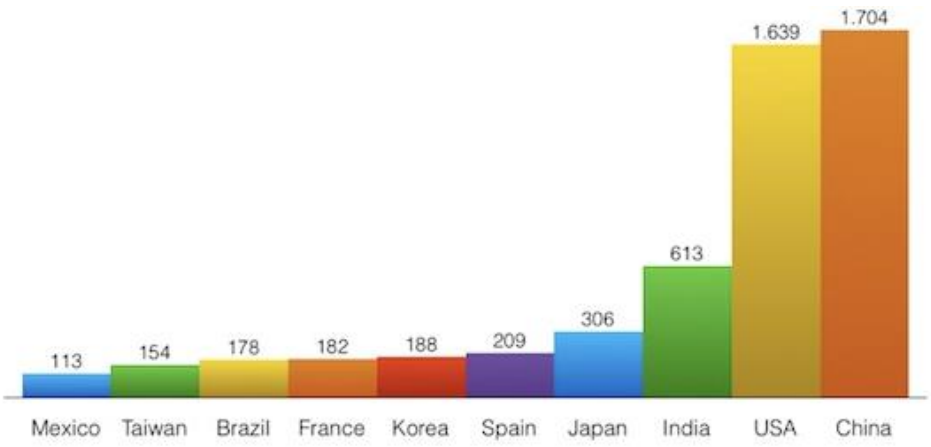


Figura 16 – Top 10 Países com avaliações do modelo CMMI - [16]

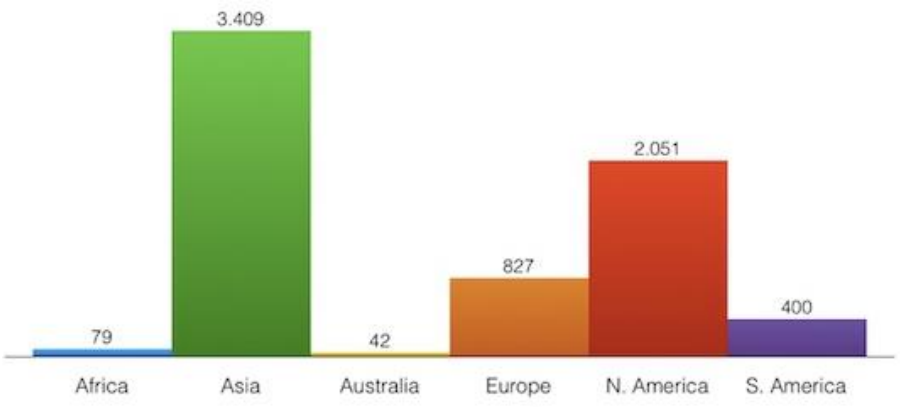


Figura 17 – Top 10 Países com avaliações do modelo CMMI - [16]

Em termos de desvantagens, podemos considerar o esforço necessário para atingir o nível 5 de maturidade. No entanto devemos ter em conta que empresas com o nível 5 são altamente processuais, com elevados padrões de qualidade e rigor e com um longo caminho até terem atingido o nível 5 de maturidade: [16]

Nível de maturidade CMMI	**Tempo médio para atingir o nível de maturidade**
Nível de maturidade número 2	Entre 4 a 11 meses
Nível de maturidade número 3	Entre 2 a 20 meses
Nível de maturidade número 4	Entre 4 a 28 meses
Nível de maturidade número 5	Entre 5 a 28 meses

No que diz respeito ao tamanho da organização para ser aplicado o modelo CMMI, por norma tende-se a cair no erro de achar que apenas grandes organizações têm possibilidade de implementar o modelo, o que não é verdade:

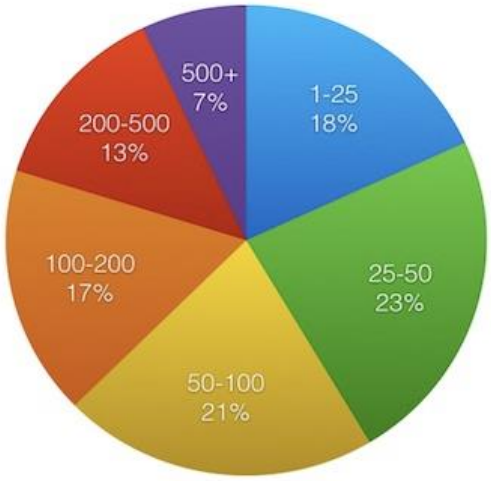


Figura 18 – Número de pessoas nas organizações com modelo CMMI - [16]

Performance Category	Median Improvement	Number of Data Points	Lowest Improvement	Highest Improvement
Cost	34%	29	3%	87%
Schedule	50%	22	2%	95%
Productivity	61%	20	11%	329%
Quality	48%	34	2%	132%
Customer Satisfaction	14%	7	-4%	55%

Figura 19 – Performance geral do modelo CMMI - [17]

Em suma, os benefícios do modelo de maturidade CMMI são solidamente comprovados por várias organizações e a vários níveis; custos, prazos, satisfação dos clientes, qualidade e produtividade:

Custos: [17]

- 33% decréscimo de custo médio para reparação de um defeito (Boeing);
- 20% redução do custo da unidade de software (Lockheed Martin);
- 15% redução na identificação de defeitos e custos de reparação (Lockheed Martin);

Tempo: [17]

- 50% redução no tempo de resposta (Boeing);
- 60% redução de trabalho de correcção após testes efectuados (Boeing);
- Aumento de 50% a 95% do número de *milestones* atingidas (General Motors).

Produtividade: [17]

- 25% a 30% aumento da produtividade num período de 3 anos (Lockheed Martin, Harris, Siemens).

Qualidade: [17]

- 50% redução dos defeitos de software (Lockheed Martin);
- Redução dos defeitos e da gravidade dos mesmos em pós-produção (JP Morgan);
- Melhoria na qualidade do código desenvolvido (Sanchez Computer Associates, Inc.).

No entanto é necessário compreender que atingir os níveis de maturidade do modelo CMMI não garantem à pior que a organização irá atingir os seus objectivos de negócio. No entanto, o CMMI disponibiliza uma metodologia poderosa para guiar a organização no caminho correcto.

2.2. Quality Management System

O Quality Management System (QMS) nasce das necessidades da organização em controlar e garantir a qualidade dos seus produtos e demonstrar evidências desse controlo, possuir um sistema central de informação, envolver todos os colaboradores nos objectivos da organização, e fazer o "*enforcement*" de acções de melhorias através de uma contínua verificação, análise e avaliação dos processos da organização. Um QMS correctamente implementado irá permitir suportar os objectivos da organização, aumentar a qualidade dos processos de produção, oferecer transparência das operações para os parceiros de negócio, aumentar a eficiência, reduzir desperdícios e garantir a satisfação global de todos os *stakeholders* da organização.

Sinteticamente a implementação de um QMS resulta na centralização da informação da organização, onde todos os colaboradores são envolvidos em ambos os objectivos de negócio e operacionais, existindo assim uma maior capacidade da organização em agir de forma rápida e sistemática sobre os resultados e progressos dos objectivos, aplicando correctamente medidas correctivas e sobretudo medidas preventivas.

Na seguinte figura é possível verificarmos a importância de um QMS nas organizações:

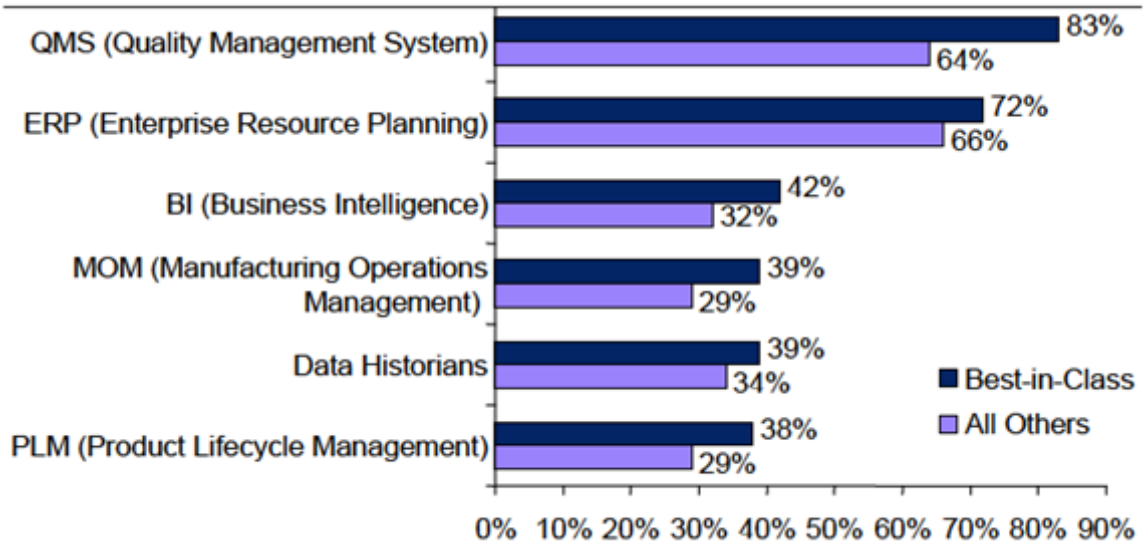


Figura 20 – Comparativo de soluções para gestão de qualidade nas organizações - [21]

Capítulo 3
Projecto de Estágio

3.1. Background

Uma das características com maior peculiaridade na RLS, enquanto uma organização de Engenharia de Software, é a extensa experiência dos seus colaboradores na Indústria e Academia, sendo especializados em Engenharia de Software com competências diversas e complementares. Apesar da RLS ainda se encontrar numa fase inicial de maturidade, o fato de ter colaboradores também eles com fortes ligações com engenharia de software, em particular com práticas de engenharia de software orientas a processos, vincula a organização com a importância e necessidade de evoluir para a gestão de processos formal.

Neste contexto foram identificadas algumas das necessidades de maior relevância para os *stakeholders* da RLS:

- **Melhoria contínua de *overhead* e *rework*** – Promover uma cultura e filosofia de melhoria contínua de forma a reduzir *overhead,* tempo gasto em *rework* e adaptar ou melhorar os processos transversais a toda a organização. Adicionalmente, estas práticas de melhoria contínua são requisito para obter os certificados de qualidade ISO 9001:2008 e CMMI, que fazem parte de objectivos estratégicos da organização para atingir objectivos chave de negócio e operacional da organização.

- **Orientação por processos** – Definir *standards* que permitam especificar acções repetíveis, convergir os outputs, normalizar actividades e consequentemente minimizar os riscos associados a actividades de gestão e operacionais.

- **Métricas** – Estabelecer uma *baseline* que ofereça a capacidade à organização para recolher métricas de processos, procedimentos e actividades, projectos e da organização como um todo. Analisar e comparar dados relacionados com projectos, tarefas e actividades, e identificar claramente as áreas para melhorias. É também importante a capacidade para conhecer e prever os estados da organização em termos de performance, através da recolha e análise de métricas, que possam ser utilizadas em cálculos de *KPIs* transversais a todas as áreas da organização.

- **Garantia de Qualidade** – Os certificados ISO 9001:2008 e CMMI são vistos como uma das necessidades mais importantes para os *stakeholders* da RLS, permitindo gerir a organização de forma objectiva e assegurar o controlo da qualidade nos seus produtos e serviços, e explorar novas oportunidades de negócio.

- **Gestão de conhecimento –** Em linha com os objectivos de estandardização e redução de *overheads* de correcção e *rework* associados com os processos de engenharia, foi também identificada pelos *stakeholders* da organização a necessidade de uma eficaz gestão de conhecimento. Desta forma a organização fica dotada dos mecanismos

necessários para uma gestão e transmissão de conhecimento sustentável (transversal a todas as áreas da empresa e com base na participação activa dos seus colaboradores).

3.2. Scope

A definição do *scope* tem como objectivo especificar que requisitos o produto actual irá contemplar e quais serão migrados para futuras *releases*. No caso de existirem requisitos descartados ou não priorizados, estes serão identificados na presente secção.

Nesta actividade foi feita a especificação do scope do *Projecto Athena*. No entanto após o levantamento de requisitos do projecto *Athena*, devido à dimensão e complexidade do projecto como um todo e sendo o estágio limitado por tempo e esforço, não seria viável abranger no estágio todo o âmbito do projecto *Athena* por completo. Por isso mesmo foi necessário definir um sub-âmbito para ser enquadrado com o estágio. Assim sendo, o scope do estágio é:

- Analisar os requisitos das normas de qualidade envolvidas (ISO9001:2008 e CMMI)
    - Estudo detalhado das suas importâncias, benefícios e requisitos.
    - Estudo da possibilidade de integrar e mapear áreas de processo entre ISO e CMMI.
    - CMMI – Estudo em detalhe da área de processo *Engineering*, nomeadamente dos processos *Verification* [ver anexo B1] e *Validation* [ver anexo C1].
- Analisar os requisitos e especificações de implementação de um QMS com conformidade com ISO9001:2008 e CMMI.
    - Estudo de diferentes mecanismos de suporte e *enforcement* do QMS.
- Especificação e configuração de um conjunto de mecanismos e ferramentas necessárias para monitorizar, controlar e garantir o cumprimento dos requisitos definidos pelo certificado de qualidade ISO9001:2008 e pelo modelo CMMI, nomeadamente dos processos (V&V) definidos.
- Analisar, identificar e levantar os requisitos de negócio e sistema de um SI.
    - Análise contextual dos requisitos e necessidades junto dos *stakeholders* da RLS.
    - Análise e levantamento dos requisitos de negócio [ver anexo H1].
    - Análise e levantamento dos requisitos de sistema [ver anexo H2].
- Contribuir para a estruturação do QMS da RLS.
    - Incluindo a definição do QM.
    - Definição do processo de Gestão Documental.
    - Especificação de *Work instructions* [ver anexo G2 e G3].
- Definir e implementar por completo os processos de Verification (VER) e Validation (VAL) de CMMI nível 3 para a RLS.
    - Definição e implementação dos processos V&V.
    - Definição e implementação dos respectivos procedimentos.
    - Definição de *checklists* utilizadas no processo de *Verification*.
        - *Code review* [ver anexo D2 e D3].
        - *Design review* [ver anexo D4].
        - *GUI review* [ver anexo D5].
        - *Requirements review* [ver anexo D6 e D7].
    - Definição de *templates*.
        - *Test Plan @ Verification* [ver anexo E6].
        - *Software Testing Report @ Validation* [ver anexo E5].

- *Requirements Traceability Matrix @ Validation* [ver anexo E4].
- *Document Review Meeting Minutes @ Verification* [ver anexo E2 e E3].

- Definição de métricas e KPI's para os processos V&V

3.3. Metodologia

A metodologia de trabalho do primeiro semestre envolveu, numa primeira fase, o estudo dos fundamentos, conceitos e estado de arte relacionados com os objectivos do estágio e da RLS. Esse estudo passou pelos princípios e factores dos quais a Qualidade está dependente, tipos diferentes de normas e modelos de Qualidade e quais os seus propósitos, e ainda um estudo da importância do QMS e como é possível através de Sistemas de Informação (SI) fazer o suporte e *enforcement* dos QMS nas organizações.

No primeiro semestre tive ainda a oportunidade de participar na especificação do QMS da RLS. A primeira premissa adoptada na elaboração do QMS foi o alinhamento com os requisitos ISO 9001:2008. Com este enquadramento foi especificado o *Quality Manual*, incluindo o Mapa de Processos da organização, a identificação dos *stakeholders* da RLS e o Organigrama da organização.

Adicionalmente tiveram de ser discutidos assuntos relacionados com o nível de formalismo pretendido pela RLS, na especificação dos seus processos, e como seriam suportadas alterações ao QMS e todos os artefactos associados (ex.: documentos no formato digital). Foi necessário discutir permissões sobre os documentos e artefactos, estados e fluxos dos artefactos, necessidade de acções de revisão, notificações de alterações/actualizações de documentos e como poderia ser possível envolver todos os colaboradores no enriquecimento documental da organização, sem exigir um elevado nível de controlo manual, normalmente associado às actividades relacionados com alterações de documentos, mas ao mesmo tempo respeitar os requisitos não só da RLS como da ISO9001:2008.

Na fase de análise de requisitos foi necessário em primeiro lugar identificar o Background da organização, que tem como objectivo fazer uma primeira introdução contextual ao negócio da organização, e especificar o Scope do projecto *Athena*. Contudo o Scope foi evoluindo em termos de detalhe à medida que foram realizados os requisitos de negócio, especificando o que faria parte ou não do projecto. Depois foi necessário realizar análises contextuais junto da RLS, para assim ser possível minimizar os erros na análise de requisitos de negócio e requisitos de sistema. Estas três análises foram necessárias não só pelo acréscimo de valor entregue ao estágio e à RLS, mas também pelo facto de minimizarem os riscos de insucesso, maximizar a capacidade de gestão e planeamento, permitir o correcto levantamento das necessidades da RLS e cumprindo assim os princípios da disciplina de Engenharia de Software. As respectivas fases de análise de requisitos serão descritas na próxima secção.

3.3.1. Análise Contextual Requisitos

A análise contextual [ver anexo – H3] envolveu numa primeira instância conhecer e compreender as actividades diárias desempenhadas pelos colaboradores da RLS. Para isso foi necessário um acompanhamento físico (presencial) das actividades dos vários tipos de colaboradores e tentar perceber as actividades que precisam para desempenhar as suas funções, e que problemas encontram durante essas actividades. Durante os períodos em que tive oportunidade de assistir ao dia a dia dos colaboradores, fui tirando notas de situações que identifiquei como possíveis problemas ou dificuldades nas suas actividades, assim como

apontamentos de comentários feitos pelos colaboradores, para que durante as entrevistas fosse possível cruzar informação da minha visão com a dos colaboradores e identificar assim as necessidades e sugestões de cada colaborador.

Depois da actividade de acompanhamento das actividades dos colaboradores, elaborei uma tabela (ver abaixo) com os *roles* da RLS e as respectivas áreas de actividade. Desta forma foi possível preparar um conjunto de perguntas adaptadas à natureza das suas actividades e perceber que diferentes *roles,* mesmo que actuando na mesma área, possuem diferentes necessidades e visões sobre as necessidades e expectativas para o projecto *Athena*.

	CEO	COO	CTO	CQO	Project Manager	Technical Manager	SW Eng.	Total
Board	X	X	X					3
Area Management	Strategic	Operations	Engineering	Quality				4
Quality Assurance	X	X	X	X	X	X	X	8
Software Development (coding/testing)						X	X	2
Technical Management						X		1
Project Management					X			1
Workers	X	X	X	X	X	X	X	8
Total	4	4	4	2	4	4	3	

A análise contextual terminou com as entrevistas aos *stakeholders,* que foram gravadas e posteriormente estruturadas em documento de texto, para que fosse possível a sua revisão e aprovação por parte dos mesmos. A análise contextual foi sem dúvida uma mais-valia para as actividades seguintes de análise de requisitos, pois permitiu obter um profundo conhecimento das actuais necessidades da RLS, e facilitar assim o levantamento de requisitos de negócio.

3.3.2. Análise de Requisitos de Negócio

Muitas das vezes a análise de requisitos de negócio [ver anexo – H1] é feita incorrectamente porque não existe uma percepção das diferenças entre requisitos de negócio e objectivos ou porque confundem requisitos com funcionalidades de produtos ou sistemas. Há vários tipos de requisitos, e os requisitos de negócio devem traduzir o que é necessário entregar ao cliente para acrescentar valor. Não é preocupação nos requisitos de negócio identificar de que forma será entregue, mas apenas o que tem de ser entregue ao cliente.

Para a análise de requisitos de negócio comecei por ler bastante sobre o assunto. Apercebi-me durante o estudo que se tratava de uma actividade bastante exigente e como tal solicitei o apoio do meu orientador da RLS. O apoio consistiu na elaboração em conjunto de um "*mind mapping*", analisando e categorizando sistematicamente a informação recolhida na fase de Análise Contextual, em 4 categorias de conhecimento para o projecto Athena:

- Gestão de Conhecimento (transversal a vários níveis desde conhecimento empírico e cientifico sobre a actividade de empresa e projectos, até conhecimento analítico operacional e organizacional)
- Gestão de Projecto
- Time and Task Tracking
- Gestão de QMS

Com este "*mind mapping"* e com a informação recolhida pelas análises contextuais, foi-me mais fácil começar a fazer o *drill down* dos tópicos para requisitos de negócio. Comecei então por descrever as motivações da RLS e de seguida a especificação dos requisitos de negócio. A especificação das motivações é importante já que na maior parte das vezes traduz directamente o que necessita existir para acrescentar valor à organização e identificar assim um requisito de negócio. [20] Quando foram identificados todos os requisitos de negócio, foi solicitado à RLS a revisão dos mesmos e também do *Scope* actualizado e já completamente definido. O *Scope* é importante ser bem definido pois é parte integrante do documento de requisitos e como tal tem de ser revisto e dado como aceite pelos *stakeholders* do projecto; Uma vez revisto e aceite o documento, ele passará para um estado *Baselined* e todas as alterações posteriores terão de passar por um processo de controlo de alterações.

3.3.3. Análise de Requisitos de Sistema

Na análise de requisitos de sistema [ver anexo – H2] pretende-se a especificação de como serão cumpridos os requisitos de negócio anteriormente especificados. Essa especificação implica assim que por cada requisito de negócio exista um ou mais requisitos de sistema. Trata-se portanto de um nível intermédio entre os requisitos de negócio e os requisitos de software.

Para a análise de requisitos de sistema tive logo de início acompanhamento por parte da RLS, onde me foi mostrado como deve ser feita a especificação de requisitos de sistema com base na definição de requisitos de negócio, assim como conceitos e dicas para a respectiva análise de requisitos de sistema. Foi ainda necessário complementar com algum estudo, mas os conhecimentos adquiridos aquando da análise de requisitos de negócio revelaram-se bastante úteis para compreender mais rapidamente a análise de requisitos de sistema.

3.3.4. Desafios Projecto

Após a análise de requisitos foi possível identificar a oportunidade de agir sobre a área de processual de desenvolvimento de software, uma vez que até à data a RLS não possuía nenhum processo definido. Tendo ainda em vista o estudo feito no primeiro semestre, eram já conhecidos os benefícios e impactos do modelo CMMI em empresas da indústria de desenvolvimento de software. Como tal foi escolhido agir sobre a área de processo *Engineering*, implementando 2 processos (V&V) base para o desenvolvimento correcto e adequado de software.

Apesar dos desafios e riscos conhecidos, a decisão foi tomada não só pelo desafio em si, mas também pelo gosto das áreas de conhecimento de melhorias continuas, qualidade e Engenharia de Software. Independentemente dos riscos associados, a oportunidade de desenvolver 2 processos, num ambiente empresarial de elevado nível rigor, completude e exigências, de grande importância para qualquer empresa de desenvolvimento de software, constituí só por si um ganho pessoal inigualável.

Nos próximos capítulos é descrito todo o percurso até à definição dos processos V&V.

Capítulo 4
4.1 QMS RedLight Software

O QMS da RLS é neste momento constituído pelo *Quality Manual* (QM) [ver anexo – A1] da organização, que inclui:

- *Stakeholders.*
- Missão, Visão, Valores e Política de Qualidade.
- Organigrama.
- Mapa de Processos, descrição e hierarquia.
- Especificação de Processos, Procedimentos, Instruções de Trabalho e Documentos de suporte.

A figura seguinte descreve a relação entre os diferentes níveis hierárquicos do QM da RLS:

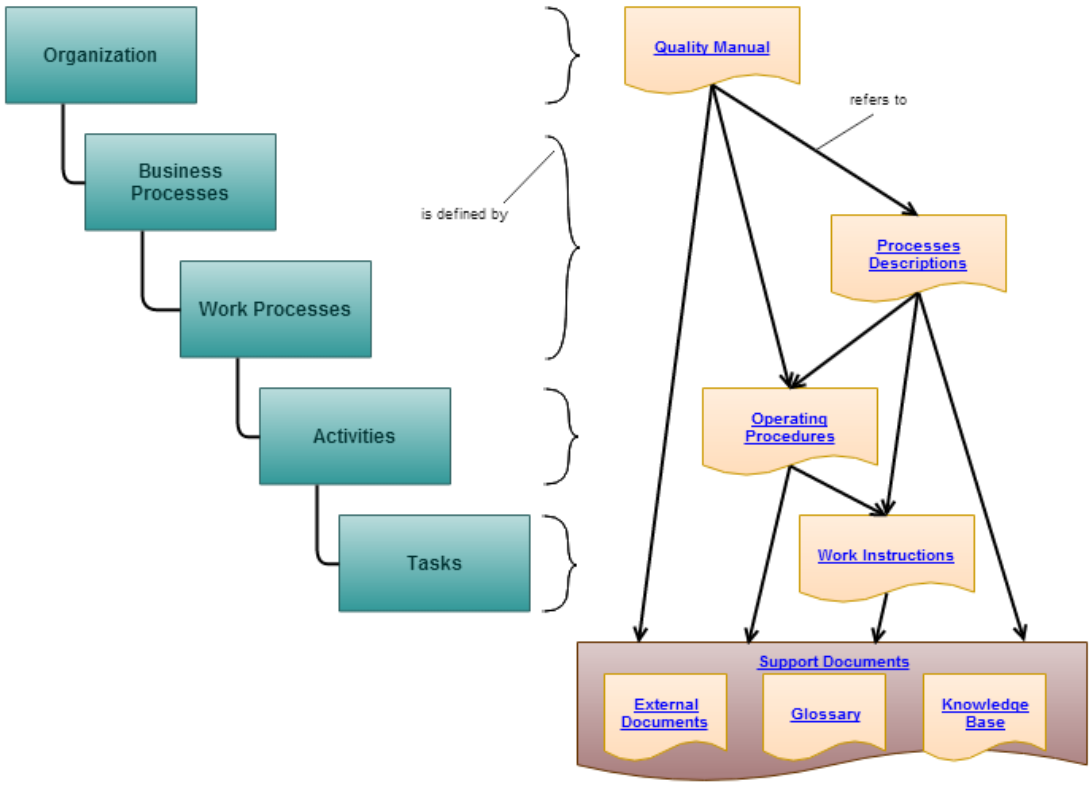


Figura 21 - Relação hierárquica entre processos e documentos

Ao nível da organização temos o próprio QM. O QM é o documento de topo, que tem como objectivo especificar a orientação eficaz do QMS, em conformidade com os requisitos ISO 9001:2008. No que diz respeito aos processos da RLS, estes são divididos em 2 níveis: *Business Processes* e *Work Processess*. Neste nível os documentos descrevem os processos, e tudo o que é necessário saber sobre os processos. O próximo nível diz respeito às actividades. As actividades são descritas pelos documentos *Operation Procedures* que descrevem como

desempenhar as actividades, e os passos necessários. O último nível da hierarquia pertence às tarefas. As tarefas são descritas pelos documentos *Work Instructions* que têm como objectivo descrever como desempenhar tarefas simples.

Observando o mapa de processos da RLS [ver anexo – A1], é possível identificar uma das áreas *core* de negócio da empresa: *Product Realization*. *Product Realization* é um *Business process* com o objectivo de criar e manter os produtos ou serviços de software, satisfazendo as necessidades e exigências dos seus clientes. Para isso o processo deverá ser suportado por um conjunto de outros processos do nível *Work Process,* de diferentes categorias: *Project Management, Engineering e Support*. Foi precisamente na categoria da *Process Area* (PA) *Engineering* que incidiu o foco do estágio, tendo sido desenvolvidos 2 processos, *Validation* e *Verification* (V&V), do modelo CMMI (ambos os processos correspondem ao nível 3 de maturidade).

4.2 Implementação QMS

A RLS tem como objectivo implementar um QMS que esteja em conformidade com os requisitos do certificado ISO 9001:2008. Ao implementar um QMS estamos a reforçar a filosofia de uma empresa orientada a processos, que pretende demonstrar a sua capacidade em satisfazer os requisitos dos seus *stakeholders* e implementar, de forma sistemática, melhorias transversais a todas as suas operações. No entanto é natural que uma empresa orientada a processos precise de definir e manter os processos, ou seja, precisa de gerir eficientemente a sua documentação. Deste modo, e tendo em conta o certificado ISO 9001:2008 onde vários requisitos dizem respeito ao controlo de documentos e registo de evidências, é natural a necessidade da empresa em possuir um sistema de gestão documental adequado. É importante referir ainda o facto de a RLS ter o objectivo de se tornar uma empresa *paperless* (sempre que possível), o que exigiu uma solução inteligente para o sistema de gestão documental, capaz de cumprir vários requisitos:

- Conformidade com os requisitos ISO 9001:2008.
- Centralização da informação - facilmente acessível a todos os colaboradores, e que seja possível editar sem a necessidade de uma autorização formal.
- *Traceability* – Incluí o registo automático do histórico de alterações por autor e data, e a sinalização das alterações efectuadas.
- Capacidade de *backups* e restaurar versões anteriores dos documentos.
- Definir restrições (ex.: ver, editar, eliminar) de acordo com os diferentes *roles* (ex.: *User*, *Editor in Chief*).
- Notificações automáticas para os colaboradores de novos/actualizados documentos.
- Definir ciclos de vida para os documentos, onde apenas os documentos publicados estão visíveis para todos os colaboradores.
- Capacidade de implementar um sistema de revisão documental, onde os documentos possuam diferentes estados (ex.: rascunho, em revisão, aprovado) e estes sejam alterados seguindo *workflows* definidos pela empresa. Os estados deverão implicar comportamentos diferentes nos documentos. Por exemplo, um documento aprovado é visível por todos os colaboradores; um documento em revisão é visível apenas pelos revisores destacados.

Através da implementação de um sistema de gestão documental que cumpra os requisitos acima especificados, espera-se que seja possível não só cumprir os requisitos ISO 9001:2008 para especificação do QMS da RLS, como também da capacidade em reduzir o *overhead* e formalismo associado à gestão documental de processos e outros documentos com altos níveis de formalismo (ex.: necessidade de manualmente os autores registarem

alterações/versões no próprio documento), contribuir para forma positiva para o meio ambiente, eliminar a necessidade de espaço físico para armazenamento de documentação, e sobretudo "tirar os processos das gavetas" e coloca-los onde todos os colaboradores possam ver, aprender e contribuir para a sua evolução.

4.3 QMS e Confluence

O *Confluence* surgiu por iniciativa da RLS, com o intuito de experimentar o uso de uma *Wiki* para o armazenamento de documentação. No entanto rapidamente se percebeu as potencialidades do *Confluence*, e começou a existir a possibilidade de se utilizar não só para gestão de documental mas ainda como plataforma de suporte e *enforcement* do QMS da RLS. A fase de Análise de Requisitos de Negócio e Sistema da RLS desempenhou um papel fundamental, para que existisse a possibilidade de mapear todos requisitos identificados com as funcionalidades oferecidas pelo *Confluence*, e tomar assim uma decisão com base em factos.

A utilização do *Confluence* para implementar o *QMS* da RLS começa por desenvolver uma filosofia de colaboração ao nível de toda a organização, onde todos os colaboradores são estimulados a contribuir, uma vez que os documentos podem ser editados a qualquer momento, eliminando a distinção entre os autores e os editores. As melhorias contínuas começam a existir verdadeiramente na organização uma vez que editar um documento passa a ser acessível e fácil, fazendo com que pequenos erros, antes considerados "aceitáveis", sejam agora imediatamente corrigidos. A segurança e os formalismos mínimos são mantidos, uma vez que cada colaborador possui um *login* para aceder ao *Confluence*, e todas as alterações feitas são registadas automaticamente com a data, hora e nome do colaborador. As alterações podem ser monitorizadas através de emails ou por notificações em tempo real na *inbox* do *Confluence*. Todas as alterações feitas podem ainda ser comparadas com versões anteriores, e no caso de existir não conformidades os documentos podem ser facilmente revertidos para versões anteriores. Possuindo a informação centralizada no *Confluence* permite que colaboradores de diferentes áreas de conhecimento possam contribuir e partilhar os seus conhecimentos não apenas no seu departamento, mas em todas as áreas da organização. O aspecto de colaboração é especialmente capturado pelo *Confluence* uma vez que adicionalmente a cada documento, existe a possibilidade de fazer comentários que são visíveis na página do documento mas que não fazem parte do documento em si (ex.: no caso do documento ser impresso. A informação passa a estar num estado "vivo" em que é actualizada frequentemente, e existe ainda a capacidade de guiar os colaboradores por fluxos de informação, ou seja, através de *links* (ex.: em palavras ou mesmo em figuras dos diagramas) os colaboradores podem facilmente "saltar" de secção em secção ou de documento em documento focando-se apenas nas informações necessárias para o momento.

4.4 Confluence e ISO 9001:2008

O *Confluence* desempenha ainda um papel fundamental no *roadmap* para atingir a certificação ISO 9001:2008. Através da configuração do *Confluence* e do respectivo *plugin Ad-Hoc Workflows* é possível cumprir a maioria dos requisitos de controlo de documentação ISO 9001:2008, nomeadamente cumprir na totalidade de 1 dos 6 procedimentos obrigatórios de ISO9001:2008, *Control of Documents* (4.2.3) [Ver Anexo – I1]**:**

- 4.2.3 a) – Os documentos seguem um ciclo de vida [ver anexo – A3] definido no *Ad-Hoc Workflows*, que exige a aprovação do documento antes de ser publicado e visível a todos os colaboradores. Os responsáveis por publicar o documento são automaticamente notificados em tempo real no *Confluence*, eliminando desperdícios de

tempo. Uma vez no estado "por publicar" o documento tem um período de 24h até ser automaticamente publicado (acção passível de ser revertida).

- 4.2.3 b) – Os documentos podem ser facilmente actualizados sempre que for necessário, ou no caso de existir um erro. Ao existir uma alteração ao documento, o estado do seu ciclo de vida é alterado automaticamente, exigindo que seja de novo aprovado antes de publicado e visível por todos os colaboradores (ex.: enquanto não for publicada a nova versão, a anterior mantém-se visível).
- 4.2.3 c) – O *Confluence* tem como funcionalidade base identificar as alterações e revisões dos documentos. Através do *Ad-hoc workflows* é ainda possível identificar claramente o estado do documento no topo da página (ex.: em revisão, publicado, etc.)
- 4.2.3 d) – O *Confluence* é facilmente acedido (com o respectivo login válido) desde que existe uma ligação à internet ou ligação na rede local da RLS. Pode inclusive ser acedido através de um dispositivo móvel.
- 4.2.3 e) – É feito de forma automática pelo *Confluence*.
- 4.2.3 f) – Ao serem incorporados no QMS da RLS (Secção *Support Documents*, *External Documents*), todos os documentos passam a estar sujeitos ao ciclo de vida definido, e consequentemente às políticas de actualizações e revisões. Podem ser decidido incluir uma cópia do documento no QMS ou simplesmente um *link* externo para o próprio documento.
- 4.2.3 g) – É assegurado de forma automática pelo *Confluence*. Os *links* não são alterados com as novas versões dos documentos, e sempre que existir uma nova versão os colaboradores são automaticamente notificados em tempo real.

Além deste procedimento obrigatório ISO9001:2008, a implementação do QMS através do *Confluence* e *JIRA,* neste caso*,* assegura ainda o controlo de registos e evidências, também um requisito ISO 9001:2008, *Control of records* (4.2.4) [ver anexo – I3]:

- 4.2.4 – O registo e controlo de evidências são assegurados pelo *Confluence* e *JIRA* pelas respectivas capacidades de *traceability (ex:* histórico de alterações)*,* definição de restrições, restauro de versões anteriores e capacidade de pesquisa imediata por *keywords* ou *tags* dos registos. As não-conformidades são registadas e controladas no *JIRA* (artefactos de código fonte) e no *Confluence (*documentação).

4.5 Ferramentas de suporte

Além do *Confluence* que é utilizado para suportar o QMS da RLS, existe um conjunto de ferramentas que são utilizadas no planeamento de projectos, *bug tracking*, apresentação de *dashboards*, recolha de métricas, etc.

4.5.1 Confluence

O Confluence é uma *Wiki*[2]. Tem como objectivo suportar e fazer o *enforcement* [3] do QMS da RLS. Além do QMS, o *Confluence* é utilizado para especificar requisitos, *mockups*, tarefas, agendas de reuniões, recolha de evidências, etc. No futuro o *Confluence* é visto como plataforma central para a completa gestão do conhecimento da RLS.

[2] Plataforma que tem como objectivo centralizar a informação, possibilitar a edição colaborativa e interligar a informação. Oferece ainda a capacidade de rapidamente efectuar pesquisas por etiquetas (*tags)* ou keywords.

[3] O *enforcement* é feito através da gestão documental automatizada, ex: ciclo de vida dos documentos, notificações para revisão documental, etc.

4.5.2 Gliffy plugin

Com este *plugin* é possível no próprio documento criar e editar diagramas, registar alterações feitas e reverter para versões anteriores e ainda inserir *links* nas figuras dos diagramas.

4.5.3 Ad-Hoc Workflows plugin

Este *plugin* está integrado com o *Confluence* e desempenha um papel fundamental na gestão documental da RLS. É através da configuração deste *plugin* que é possível notificar os colaboradores quando existe uma alteração/actualização nos documentos, definir os ciclos de vida dos documentos, garantir que apenas os documentos aprovados são publicados, etc.

4.5.4 Kwik plugin

Este *plugin* foi integrado no *Confluence* e é actualmente utilizado nas actividades de inspecções de documentação com o objectivo de registar não-conformidades encontradas nos documentos. Tem como vantagem ser possível identificar a não-conformidade directamente no local do documento.

4.5.5 JIRA

O *JIRA* é uma ferramenta avançada de planeamento e *tracking* de projectos. Neste momento é utilizado como *bug-tracker* e planeamento dos projectos (é totalmente compatível com metodologias ágeis, SCRUM e Kanban). Está integrado com o *Confluence* para que seja possível criar tarefas recursivas (issues) no calendário (ex.: 3 em 3 meses verificar se os documentos externos foram alterados), registo de não-conformidades e submissão de possíveis melhorias para a RLS. As melhorias são inseridas como *issues* do tipo QMS *Issues*.

4.5.6 GITHub

Ferramenta utilizada como repositório para o controlo de versões do software em desenvolvimento pela equipa.

4.6 Conclusão

A implementação do QMS através do *Confluence* tem-se mostrado uma mais-valia na simplificação e automatização de alguns requisitos e procedimentos ISO 9001:2008, e nos objectivos de reforçar a filosofia de colaboração e *process-driven* da RLS. No entanto pelo meio do processo de implementação foram investidas inúmeras horas de pesquisas para soluções elegantes de funcionalidades que não estão incluídas por *default* no *Confluence* e no *JIRA*. Ao utilizar o *Confluence* a maior desvantagem diz respeito ao formalismo, nomeadamente formatações de conteúdo e necessidade de exportar alguns documentos para o *pdf/word (*ex.: necessidades dos clientes), onde pelo meio se perde alguma capacidade de formatação formal. No entanto a acredita-se que o verdadeiro valor está no propósito e no conteúdo, e que será uma questão de tempo até que este tipo de implementações seja uniforme.

No próximo capítulo é descrito todo o processo necessário até à definição e implementação dos processos V&V.

Capítulo 5
5.1 Processos

Os processos têm como propósito documentar "o que tem de ser feito" e "como fazê-lo" para que seja possível repetir de forma consistente o sucesso de acções ou acontecimentos anteriores. O processo é definido com base nas acções e acontecimentos de sucessos comprovados, e adaptado ao contexto, objectivos e necessidades da empresa. Tendo como exemplo um caso informal de culinária: Ao fazermos um bolo através do uso de uma receia, estamos a utilizar um conjunto de ingredientes (ex.:*inputs*), condições de temperatura e tempo (ex.:*entry and exit conditions*) e os passos de como juntar tudo para cozinhar o bolo (*procedures*). Se gostarmos do bolo, certamente que iremos querer guardar a receita para uma próxima vez; A receita neste caso pode ser vista, de forma informal uma vez mais, como um processo que alguém documentou quando cozinhou com sucesso o bolo.

"*Isto significa que uma empresa ao adoptar processos de outras empresas, de sucesso mundial, vai conseguir atingir o mesmo sucesso?*" Dificilmente. Em primeiro lugar é comum as "receitas" de cada empresa serem secretas. Em segundo lugar o facto ter acesso a receitas de culinária de um *Chef*, não faz da pessoa automaticamente um *Chef Michelin*, não é verdade?

Um processo deverá ser conciso, bem estruturado, desenhado para ser dinâmico e derivado das políticas, *standards*, objectivos e necessidades da própria empresa. Ao definir um processo é necessário respeitar os parâmetros definidos pela ISO e CMMI. A estrutura definida para todos os processos da RLS foi **[48]**:

- *Purpose* – Porque é desempenhado o processo?
- *Objectives* – O que se pretende alcançar e quando?
- *Roles* – Quem faz é responsável por qual actividade?
- *Inputs* [4] – Quais os artefactos usados no processo?
- *Outputs* – Quais os artefactos produzidos pelo processo?
- *Start events* – Quais os eventos que definem o início do processo?
- *End events* - Quais os eventos que definem o fim do processo?
- *Entry criteria* – Quais as condições que necessitam de ser cumpridas antes de iniciar o processo?
- *Exit criteria* – Quais as condições que necessitam de ser cumpridas antes de terminar o processo?
- *Context* – Onde é desempenhado o processo (ex.: relação com outros processos, hierarquia, mapa de processos, etc.)?
- *Flow* – Qual o fluxo das actividades do processo?
- *Procedures* [5] – Quais os procedimentos que fazem parte do processo?

[4] Os *inputs* podem ser alterados/modificados e tornarem-se *outputs*.
[5] Os Procedimentos especificam os passos necessários para desempenhar um conjunto de actividades.

- *Monitoring* – Como é medida a performance do processo?

Os processos podem ainda ser definidos de várias maneiras:

- Gráfica –
    - Flowcharts.
    - *Cross-Functional Diagram* ou *Swim Lane Diagram.*
- Texto –
    - *ETVX (Entry – Task – Validation – Exit).*

Não existindo um formato obrigatório para a definição de processos, deve-se procurar sempre um equilíbrio entre texto e figuras. Nos processos V&V definidos para a RLS foi procurado utilizar sempre que possível figuras para representar informação essencial, complementadas por texto com os detalhes que não foram possíveis de incluir.

5.2 Métricas e Key Performance Indicators

Frases como "*o que é medido é gerido*", ou "*o que não é medido é esquecido*" **[47]** são comuns na literatura de Gestão e não são apenas frases. São factos. Neste contexto a utilização de métricas e *Key Performance Indicators (KPIs)* é essencial, especialmente para uma empresa orientada a processos, com objectivos de melhorias contínuas.

Uma métrica é uma medida transformada num *standard*. Por exemplo, medir o tempo gasto em tarefas/actividades é obrigatório, como tal é uma métrica da empresa. No entanto uma métrica apenas tem pouco significado, sendo necessário relacionar uma ou mais métricas num contexto válido, para que seja possível obter dados que permitam comparar a performance ao longo do tempo. Ao relacionar duas ou mais métricas com os objectivos definidos, temos indicadores de performance (*KPIs*). Os *KPIs* têm como objectivo quantificar performance e progresso no contexto de negócio e objectivos da empresa. A metodologia na definição de métricas e *KPIs* consiste:

1. Identificar as áreas de processo chave para a empresa.
2. Definir objectivos.
3. Definir as métricas a necessárias.
4. Identificar e definir os *KPIs*.

Uma vez definidos os *KPIs*, estes oferecem a capacidade de avaliar continuamente a performance e o progresso dos objectivos definidos pela empresa. No entanto as métricas e os *KPIs* devem ser escolhidos cuidadosamente, uma vez que demasiadas métricas causam *overhead* (na maioria dos casos as métricas necessitam de ser registadas e recolhidas manualmente) nas actividades da empresa, e demasiados *KPIs* tendem a dispersar a atenção dos verdadeiros objectivos. Por outro lado a escolha insuficiente de métricas, ou a escolha pelo facto de serem fáceis de recolher, não contribuí para a capacidade de análise da performance dos objectivos, acabando por ser abandonadas, causando desperdício de tempo e frustração nos colaboradores.

Para que seja possível definir com sucesso as métricas e os *KPIs* relevantes para a empresa é necessário ter conhecimento não só dos objectivos da empresa, como da sua visão e estratégia [Figura 1 – Metrics and KPIs]. As métricas e *KPIs* dos processos V&V foram definidas depois de ter sido estudado o *Quality Manual* da RLS e ter sido feita uma análise e levantamento completo dos requisitos de negócio junto dos *stakeholders* da RLS.

Figura 2 - Metrics and KPIs, [27]

A utilização das métricas e *KPIs* definidos tem objectivos como:

- Aumentar a capacidade de estimar de forma precisa actividades e tarefas a desempenhar – Através do registo de métricas como o tempo gasto, ou o número de erros [6] encontrados por tamanho/complexidade do artefacto, será possível tomar decisões com base em acontecimentos anteriores (factos).
- Monitorizar o progresso – Métricas como o número de erros encontrados, uma vez comparadas diariamente/semanalmente, por exemplo, poderão indicar a maturidade do artefacto e consequentemente a possibilidade de estarem prontos para avançar para próxima fase de desenvolvimento.
- Melhorar os processos – Utilizando métricas que indiquem as fases onde os erros encontrados foram introduzidos, poderão indicar áreas de processos onde existem problemas e a necessidade acções correctivas. Por exemplo, se houver uma grande quantidade de erros introduzidos na fase de desenvolvimento do código, poderá indicar que os *developers* não tiveram formação suficiente.
- Priorização de acções – Métricas que indiquem o número de erros encontrados por *man-hour,* por exemplo, dão a possibilidade de optar por métodos mais eficientes em situações onde exista falta de recursos (*walkthrough vs inspection*, ou *black box vs white box*).

[6] Na RLS os erros são classificados como *defects* ou *issues*. Sendo que *defects* são os tipos de erros encontrados durante o processo de validação (VAL) e *issues* são erros encontrados durante o processo de verificação (VER). No entanto no relatório, de modo a simplificar a leitura, é sempre usada a mesma nomenclatura: erros.

5.3 Verification and Validation

Software *Verification* [ver anexo – B1] e Software *Validation* [ver anexo – C1] são dois processos de qualidade de Engenharia de Software que visam aumentar a capacidade das empresas em medir, avaliar e aumentar a qualidade do software por elas produzido. Não devem ser confundidos com processos de *Quality Assurance*; são processos complementares mas não iguais. Estando estes dois processos tão intrinsecamente ligados, é comum fazer-se referência aos mesmos por V&V (*Verification and Validation*). De um modo geral os objectivos do processo *Verification* são assegurar que o software está de acordo com as suas especificações técnicas e em correcto funcionamento, enquanto o processo de *Validation* visa garantir que o software corresponde de forma completa às necessidades dos *stakeholders (*ex.: clientes, utilizadores, etc.). Note-se como um conceito (verification) é centrado nas actividades e artefactos técnicos enquanto o outro (*validation*) faz a ponte com o contexto real de aplicação do software. É comum encontrar na literatura as seguintes caracterizações informais dos processos:

- *Verification: Are we building the product right?*
- *Validation: Are we building the right product?*

Os processos V&V foram desenhados para serem aplicados de forma conjunta e contínua em todo o processo de desenvolvimento de software. Começam, inclusive, a ser aplicados antes de existir um projecto de trabalho, com a Verificação a iniciar-se com o desenvolvimento da proposta de trabalho a ser enviada à parte interessada (ex.: Cliente). É também possível que os processos V&V se estendam até à fase de manutenção do software, embora em situações comuns e onde não existe a necessidade de manutenção contínua, os processos de V&V iniciam-se com a fase de análise de requisitos e terminam com o encerramento do projecto. No modelo CMMI-DEV ambos os processos correspondem ao nível 3 de maturidade, da área de processos *Engineering*.

O processo de *Verification* envolve a análise estática dos artefactos que são produzidos ao longo de todo o processo de desenvolvimento de software, nomeadamente o código fonte, documentos de Design e Arquitectura, documentos com análise de Requisitos, etc. Este processo consiste em inspecções (*peer reviews*) nos quais os revisores verificam se as especificações estão correctas, se os padrões de qualidade correspondem aos exigidos e se não existem erros. Trata-se de um processo com um custo/esforço considerável em prol da qualidade dos artefactos e consequentemente do software produzido, uma vez que as inspecções são feitas de forma manual, onde os revisores analisam linha a linha – (ou página a página) o artefacto, exigindo sempre a participação de pelo menos 2 colaboradores. Embora o custo (o custo para este caso, na RLS, é medido directamente em *man-hours*) necessário para encontrar erros seja alto, o custo para corrigir os mesmos é baixo uma vez que expectavelmente os erros são detectados nas fases inicias de especificação e de desenvolvimento.

O processo de *Validation* corresponde à técnica de análise dinâmica dos artefactos (neste processo os artefactos considerados são apenas o código fonte), que consiste em executar o software para validar o seu comportamento. O comportamento do software é avaliado conforme os resultados esperados dado determinado *input*. Para que o comportamento seja

considerado correcto, o software tem corresponder simultaneamente às especificações funcionais e não funcionais (ex: *performance*). O processo de *Validation* consiste assim aplicar técnicas de teste do software (*software testing*), onde o objectivo é demonstrar que o software cumpre os requisitos, especificações e as necessidades dos *stakeholders*, aumentando a confiança na qualidade do software, e reduzindo o risco deste falhar quando estiver a desempenhar as suas funções em ambientes reais. É importante referir que o processo de *Validation* não irá garantir a ausência de erros, mas sim identificá-los durante as respectivas actividades. Este processo tem também um custo elevado, não só na identificação de erros como também na correcção dos mesmos (embora o custo para corrigir um erro aumente, significativamente, nas fases mais avançadas onde é identificado), pelo que deverá ser sempre um compromisso entre o orçamento, tempo e qualidade desejada. Embora a qualidade não possa ser garantida pelo processo, este contribui de forma bastante significativa para a sua melhoria.

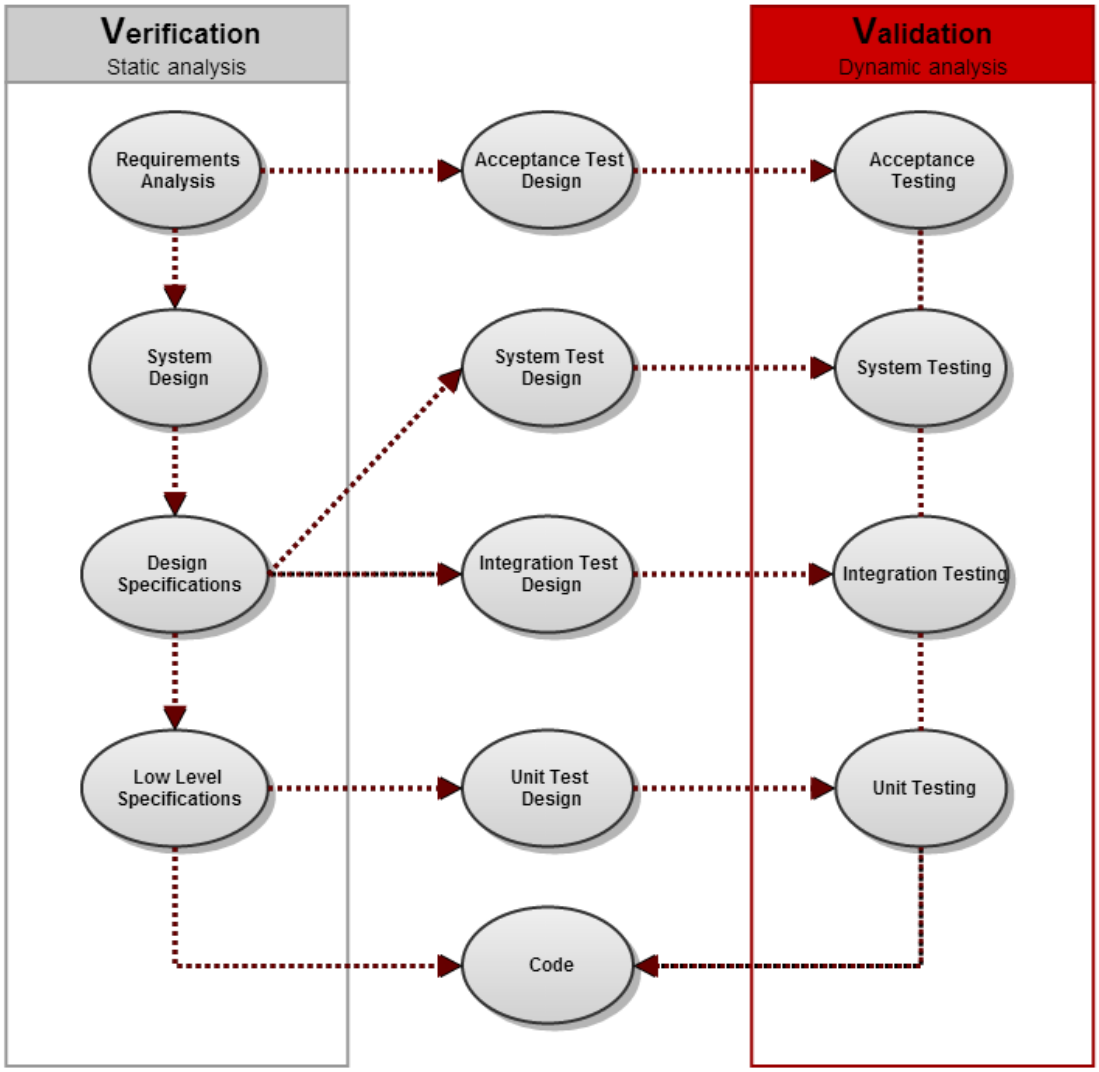


Figura 22 - Relação V&V

Os processos de V&V estão directamente ligados à qualidade dos artefactos produzidos, mas têm também influência em outros factores de grande importância não só para o processo de desenvolvimento de software, como também para a própria imagem da empresa. Podemos destacar como principais benefícios:

- Os erros são detectados nas fases iniciais, o que permite recorrer a soluções reflectidas, ao invés de soluções rápidas, para a correcção dos erros, reduzindo assim custos e *rework* durante o desenvolvimento de software.
- Gera evidências de que o software desenvolvido está em conformidade com as necessidades dos *stakeholders* e que respeita os padrões de qualidade.
- Cria uma cultura onde a qualidade dos artefactos é constantemente medida e avaliada, fazendo com que os colaboradores sejam consistentes na qualidade do trabalho que produzem. É especialmente importante para os colaboradores menos experientes, ou novos na empresa.
- Os processos de V&V são um requisito em determinadas áreas de negócio como a saúde, governo, aeroespacial, etc. Abre assim novos mercados à empresa.

De facto, os processos de V&V desempenham um papel tão importante no desenvolvimento de software que existem empresas especializadas em *outsourcing* de *Independent* V&V. A própria comunidade em torno do modelo CMMI começa a sugerir que empresas que procuram a certificação de CMMI L2 optem por fazer um esforço maior e implementem *CMMI L2* + V&V (recorde-se que os processos V&V pertencem ao nível 3 de maturidade do modelo CMMI) [23, 24], uma vez que os V&V são processos essenciais para este tipo de empresas especializadas e portanto deveriam pertencer ao nível 2 do modelo.

No próximo capítulo é explicada toda a especificação do processo *Verification*.

Capítulo 6
6.1 Verification Process

O processo de *Verification* [ver anexo – B1] da RLS é constituído por apenas um procedimento [ver anexo – B2], que especifica os detalhes necessários para desempenhar as inspecções que fazem parte do processo. O documento do processo especifica os critérios de como o processo é inicializado, terminado, desempenhado, controlado e medido. O documento [ver anexo – B1] começa com a identificação do mesmo onde é incluído o *Process Owner*, a data e informação relativa próprio documento em si:

- Tabela de acessos, aprovações e revisões.
- Tabela com os documentos e referências aplicáveis ao processo.
- Tabela de conteúdos.

Na secção 2 *Introduction* é feita uma introdução ao processo e definido o seu propósito, objectivos, *Outcomes*, audiência e acrónimos e definições usados no documento. Sobre esta secção é importante referir que os *Outcomes* foram definidos pela RLS.

A secção 3 *Process Overview* começa com uma figura que pretende resumir todo o *workflow* do processo, onde é possível ver:

- Os *inputs* e *outputs* do processo.
- A relação entre processos e procedimentos (*Documents Review*)
- As iterações do processo nos artefactos e as respectivas melhorias ao processo.

De seguida é especificado o evento que determina quando o processo de *Verification* se inicia e termina:

Start Event – sempre que um artefacto é criado ou actualizado.

End Event – termina com o desempenho de uma inspecção.

É importante referir que não é possível desempenhar inspecções formais em todos os artefactos; consequentemente, embora todos os artefactos estejam sujeitos a aprovação, alguns deles poderão ser aprovados com uma inspecção aligeirada [7](ex.: *Ad-Hoc*). Ainda assim os artefactos que necessitam de ser aprovados com uma inspecção formal, podem ser revistos várias vezes através de inspecções *lightweight* e só quando forem submetidos para aprovação, serem submetidos a uma inspecção formal. O *End Event* está associado ao desempenho das inspecções, que só terminam se cumprirem as condições de "saída" descritas no respectivo procedimento [ver anexo – B2].

Especificados os eventos que determinam o início e fim do processo, é agora necessário especificar as condições de entrada e saída do mesmo. Estás condições têm obrigatoriamente de ser cumpridas para que o processo possa ser iniciado e dado como terminado. Uma das condições de entrada para o processo é o número de colaboradores disponíveis para

[7] Uma inspecção aligeirada significa uma inspecção do tipo informal, onde não é obrigatório reunir com vários colaboradores, nem exigido o relatório com evidências (ex.: erros encontrados, métricas) da inspecção.

desempenhar uma inspecção. Sendo que cada inspecção específica o número mínimo de colaboradores necessários temos:

- 1 elemento para *Ad-hoc*.
- 3 elementos para uma *Walkthrough*.
- 4 elementos para *Inspection*.

Esta condição de entrada e o número de elementos por cada tipo de inspecção, foram especificados para que seja possível adaptar o processo ao tamanho do *staff* actualmente disponível na RLS, uma vez que de momento o *staff* é inferior a 10 colaboradores, e alguns desses colaboradores encontram-se distribuídos por diferentes localizações geográficas. No entanto os tipos de inspecções foram escolhidos não só com base no tamanho do *staff* da RLS, mas também de acordo com os objectivos e necessidades que cada tipo de inspecção oferece, como iremos ver mais à frente no respectivo procedimento *Documents Review*.

Na secção 3.4 e 3.5 (figura 2 - *RedLight Software processes map* e figura 3 – V&V *overview* respectivamente) [ver anexo – B1] têm como objectivo localizar o processo no mapa de processos da RLS (figura 2) e ilustrar a relação entre as actividades dos processos V&V (figura 3). Na figura 3 podemos verificar as várias actividades do processo de desenvolvimento de software sujeitas aos processos de V&V, e a respectiva relação entre elas. No processo *Verification* a orientação vertical das setas, de cima para baixo, pretende mostrar que cada actividade terá de ser verificada para que seja possível iniciar a próxima actividade. Estas actividades pertencem ao processo de desenvolvimento de software onde os objectivos de cada uma são:

- *Requirements Analysis* – especifica o que software deve fazer (requisitos funcionais), e por vezes o que não deve fazer também, e as restrições associadas (requisitos não-funcionais) em como o fazer caso existam. A análise de requisitos engloba os documentos de *BRS*, *YRS* e *SRS*.
- *System Design* – especificação de alto nível do sistema onde é feita a descrição do design da arquitectura que deverá implementar os requisitos identificados.
- *Design Specifications* – especificação em detalhe da arquitectura onde são descritos os módulos e componentes do sistema e a relação entre eles. Nesta fase deverão ser criados os casos de teste ao nível de integração dos módulos e do sistema já integrado (*Integration* e *System Testing*).
- *Low Level Specifications* – especificação de baixo nível mais próxima possível do código fonte, onde cada unidade de código é especificada no que diz respeito à sua implementação, ex.: tabelas e campos de base de dados, tipos de dados utilizados, algoritmos, etc.

Na coluna da direita, *Validation,* apenas são representadas as actividades disponíveis para validação do código (*code*), sem orientação por setas, uma vez que não é obrigatório o desempenho de todas as actividades (*Unit, Integration, System e Acceptance*) durante o processo de *Validation* [ver anexo – C1]. Estas actividades pertencem ao procedimento de *Test Execution* [ver anexo – C1], que serão detalhadas no capítulo *Validation*. Por último a relação entre as actividades dos processos *V&V* é representada na "coluna" central pelas actividades de *Test Design*. As actividades de *Test Design* correspondem ao procedimento de *Test Design and Implementation* do processo *Validation* e têm como objectivo a criação de casos de teste que permitam a validação do software. As actividades de *Test Design* estão assim intrinsecamente relacionadas com as actividades de especificação do software, verificadas pelo processo

Verification, uma vez que os casos de teste devem ser sempre derivados das especificações e mapeados para as mesmas (técnica comum na utilização de uma *Traceability Matrix* [ver anexo – E4]. Desta forma a validação do software é útil se, e só se, existir em primeiro lugar uma correcta especificação do software, que se pretende que seja garantida com o processo de *Verification*. O balão "Code" na "coluna" central representa o software desenvolvido, suportado pelos processos de V&V.

6.2 Artefactos para as Inspecções

De seguida são definidos os *inputs* e *outputs* do processo. Como *input* para o processo temos:

- *Checklists* - são documentos com o propósito de assistir os revisores durante as inspecções, chamando atenção para factores que possam indicar o estado dos artefactos. Esses factores são altamente focados em *standards*, factores de qualidade e erros comuns, melhorando com a experiência da equipa e da empresa; aumentam assim a capacidade dos revisores em não deixar passar em "claro" erros ou inconformidades.

- *Document Review Report – Template* para ser utilizado pelos revisores na identificação de erros. Tem como objectivo caracterizar os erros identificados por cada revisor.

- *Document Review Meeting Minutes (Moderators) – Template* do relatório a ser preenchida pelo moderador. Numa primeira instância servirá de informação aos revisores com o local e datas da inspecção, scope da inspecção, lista de artefactos e materiais de suporte disponíveis, equipa escolhida, etc.

- *Work Products* – são os artefactos submetidos para inspecção.

O processo *Verification* "produz" (*Output*):

- *Approved Work Products* – Artefactos verificados e aprovados.

- *Document Review Meeting Minutes* - Relatório complementado com evidências das inspecções tais como métricas, erros encontrados, notas e tarefas desempenhadas.

A decisão para o processo produzir "apenas" 2 *outputs*, reflecte-se na necessidade de reduzir a documentação produzida e centralizar o máximo possível a informação, sem significativa perda de eficiência. O relatório DRMM (*Documents Review Meeting Minutes)* [ver anexo – E2] foi especificado para que possa registar toda a informação necessária num só documento. Esta opção foi tomada tendo em conta o facto de ser usado o *Confluence* para gestão documental, uma vez que através do uso *labels*, pesquisa por *keywords*, categorização de espaços e hiperligações, é possível encontrar facilmente a informação pretendida.

6.3 Monitoring

A secção 5 de *Monitoring* descreve as métricas e KPIs escolhidas para medir, controlar e avaliar o processo. Só através da recolha de métricas será possível quantificar o sucesso do processo e identificar oportunidades de melhorias. As métricas são recolhidas durante as várias fases das inspecções (*Planning, Preparation, Meeting e Follow-up*) pelos respectivos responsáveis e registadas no relatório DRMM da inspecção. As métricas serão depois utilizadas nos KPIs:

1. Velocidade de revisão dos artefactos (medidas em hh:mm).

2. Velocidade dos revisores em encontrar erros (medidas em hh:mm/erro encontrado).

3. Quantidade de erros encontrados em determinado tamanho do artefacto (#erros/#LOC ou #páginas).

4. Esforço despendido nas inspecções (hh:mm).

Os KPIs 1,2 e 3 têm como objectivo oferecer à RLS a capacidade de determinar e controlar o tamanho ideal dos artefactos (ou de segmentos/parcelas de artefactos) a serem revistos e a velocidade de revisão nas inspecções. Actualmente estão definidas como melhores práticas que o tamanho ideal dos artefactos será entre 200 e 400 LOC (código) ou 4 a 8 páginas (documentação) por cada sessão, e deverá ser esperada uma velocidade de revisão por parte dos colaboradores entre 300 a 500 LOC ou 6 a 9 páginas. Estas práticas foram definidas para que seja possível maximizar o número de erros encontrados por cada hora de trabalho. Apesar destas melhores práticas terem sido definidas depois de uma vasta leitura de estudos e artigos sobre as práticas de empresas com resultados de sucesso na implementação de inspecções [*cisco peer code review study*], não significa tenham o mesmo sucesso na RLS e portanto os KPIs recolhidos serão indispensáveis para a implementação dos valores mais adequados à RLS, usando um processo de melhoria contínua.

Novamente com visão sobre o futuro da RLS onde está planeado a existência *dashboards* com KPIs transversais às várias áreas de operação da empresa, foram definidos 3 KPIs adicionais que têm como objectivo oferecer uma visão imediata sobre o impacto do processo Verification. Assim o KPI de *Team Effectiveness* tem como objectivo contribuir para o menor custo possível do processo mantendo a performance. O *Reviews Effectiveness* pretende avaliar até que ponto uma inspecção menos formal consegue ser produtiva em comparação com uma inspecção formal, e claramente de maior custo para a empresa. Este KPI inclui apenas o tempo das reuniões e deverá ser separado por *Walkthroughs* e *Inspections* para que seja possível a comparação. Por ser excluído o tempo de preparação da *Inspection* em caso de resultados semelhantes a *Inspection* ficará em clara desvantagem. Este KPI desperta bastante curiosidade de análise uma vez que várias empresas reportam que inspecções *lightweight* são tão ou mais produtivas que inspecções formais. O KPI de *Reviews Total Effort* tem como objectivo avaliar de forma directa o custo total vs Nº de erros encontrados do processo na empresa. Estes 3 KPIs são ainda medidos contra o Nº total de erros encontrados pelas inspecções, onde são ainda utilizados pesos para distinguir a importância e impacto dos erros encontrados.

6.4 Tailoring

Na secção 6 *Process Tailoring* de alteração/adaptação do processo foi definido que não será permitido qualquer tipo de alteração. Esta decisão foi tomada tendo em conta que o processo em si já está adaptado de acordo com diferentes necessidades: esforço, tamanho da equipa, objectivos, tipo de artefactos e tipo de inspecções.

6.5 Documents Review Procedure

Esta secção diz respeito ao procedimento de *Documents Review* [ver anexo – B2], que tem como objectivo detalhar as actividades necessárias para desempenhar o processo de *Verification*. O procedimento *Documents Review* começa com a especificação de uma tabela *Overview* que sumariza toda a informação essencial, de modo a facilitar a consultar do procedimento por parte dos colaboradores da empresa. Da tabela destacam-se os seguintes campos:

- Objectivo das inspecções - onde é destacado que se pretende verificar e avaliar a qualidade dos artefactos, e não avaliar os respectivos autores.

- Condições de entrada e saída do procedimento - que têm obrigatoriamente de ser todas cumpridas de modo a evitar inconsistências, tais como o agendamento de inspecções quando os artefactos ainda não se encontram devidamente preparados para serem sujeitos a uma inspecção.

- *Inputs* e *outputs* – que especifica quais os artefactos que estão à disposição da equipa para desempenharem com sucesso a actividade de inspecção, e o que se pretende que seja produzido pelo desempenho dessa mesma actividade.

- Métricas –para medir, controlar e avaliar todo o processo.

- Guia de melhores práticas – para que seja possível tirar o maior partido da performance dos revisores e consequentemente das inspecções.

De seguida são especificados os critérios para a selecção do tipo de inspecção a desempenhar de acordo com o tipo de artefacto, objectivos da inspecção e riscos. Estes critérios são especialmente importantes uma vez que não é possível submeter todos os artefactos a inspecções formais[8], e portanto é necessário escolher de forma acertada quais os artefactos que deverão estar sujeitos a inspecções do tipo *Walkthroughs* e/ou *Inspections*. Assim para a RLS foram especificados 3 tipos diferentes de inspecções, que variam da menos formal até à mais formal [ver anexo – B2, tabela review type]:

- *Ad-hoc* – Não existem registos das actividades, nem dos erros encontrados. É aconselhável ser uma revisão individual (1 revisor) e a reunião é opcional. Neste tipo de inspecções o autor deverá escolher um revisor experiente, e é esperado que a inspecção seja usada maioritariamente utilizada para resolver dúvidas e problemas já encontrados pelo autor. Embora as *Ad-Hoc* não sejam controladas, o autor é responsável por desempenhar a inspecção conforme o procedimento *Documents review*.

- *Walkthrough* – O autor é responsável por gerir e conduzir a inspecção, desempenhando o papel de moderador durante a reunião. O autor é também responsável por preencher o relatório *DRMM*. A actividade de *rework* realizada após a reunião deverá ser verificada pelo autor, ou em conjunto pela equipa, antes de a inspecção ser finalizada e o relatório *DRMM* concluído.

- *Inspection* – Trata-se do tipo de inspecção mais formal na RLS. Existe um moderador responsável por planear a *Inspection*, que incluí a escolher as datas e a equipa, e distribuir os materiais (não só os artefactos como *checklists* e outros materiais de suporte) pelos revisores. Os revisores têm então um período de preparação individual, que antecede a reunião de inspecção. Durante a reunião se não existir um colaborador com a função de *recorder,* deverá ser o moderador o responsável por registar os erros encontrados e as evidências da inspecção (ex.: presenças, notas, métricas, etc.) no relatório *DRMM*. Após a actividade de *rework* o moderador, em conjunto com o autor ou equipa, é responsável por verificar se os erros encontrados foram resolvidos. Caso ache necessário, o moderador tem a capacidade de agendar uma nova inspecção (os critérios para agendar uma nova inspecção foram especificados no procedimento).

8 Formais significa que seguem um procedimento mais complexo, pré-definido, requerem recolha de métricas ou registos, e consequentemente têm um maior custo para a empresa.

De forma genérica podemos definir que as *Ad-Hoc* e *Walkthrough* são orientadas às necessidades dos autores, e a *Inspection* é orientada às necessidades da empresa (ex.: Padrões de qualidade). Estes três tipos de inspecções foram escolhidos de acordo com a maturidade e recursos actuais (humanos maioritariamente) da RLS. A *Ad-Hoc* não exige qualquer tipo de registo de evidências pois tem como objectivo ser o mais *lightweight* possível. Além de que deverá ser usada não só para encontrar erros, como também para encontrar soluções. Ao contrário das restantes inspecções que são utilizadas apenas para encontrar erros. É verdade que a recolha de métricas, pelo menos do número de erros encontrados, poderia ser útil para efeitos de comparação com os restantes tipos de inspecções, mas as comparações não seriam totalmente válidas uma vez que os objectivos da *Ad-Hoc* variam em relação à *Walkthrough e Inspection.*

A *Walkthrough* tem como propósito ser uma inspecção que se posiciona entre os objectivos e custos de uma *Inspection*. Comparada com uma *Inspection* trata-se de uma inspecção com menores custos por não possuir uma fase de preparação individual de cada revisor, por exigir um menor número de colaboradores já que o autor assume o papel de moderador e não existe um *recorder*. Por outro lado poderá não garantir os mesmos padrões de qualidade na medida em que sendo o moderador também o autor do trabalho, este poderá conduzir a reunião conforme entender nos aspectos que, na sua opinião, são mais importantes para revisão. Podendo assim permanecer parte dos artefactos por rever e consequentemente erros ou incumprimento de *standards* por identificar.

A *Inspection* foi especificada com base na conhecida técnica *Fagan Inspection* **[22],** criada por Michael Fagan no início dos anos 70. A *Fagan Inspection* é uma técnica de inspecção bastante rigorosa, formal e vastamente utilizada em Engenharia de Software. Uma Fagan *Inspection,* devido ao seu custo, só deverá ocorrer em artefactos que sejam considerados como críticos, completos e prontos a serem utilizados. Na literatura é possível encontrar várias fontes que indicam melhorias bastante significativas em relação à qualidade, erros produzidos e taxas de *rework,* e consequentemente redução de custos depois da implementação desta técnica nos processos das empresas **[26, 25, 27].** No entanto a *Inspection* especificada para a RLS possuí diferenças em relação ao procedimento original, que tem como objectivo diminuir os custos da *Fagan Inspection* mantendo os seus benefícios. A diferença está na fase de *overview* onde foi tornada opcional a reunião de introdução aos artefactos em revisão (embora o moderador continue a ter a capacidade de convocar essa reunião, como preparação), ao contrário do especificado na técnica de *Fagan Inspection* em que ela é mandatária. Esta alteração tem em conta não só o custo da reunião de *overview,* como também o facto de ser necessário que todos os colaboradores destacados estejam disponíveis para essa reunião. O risco dos revisores não estarem dentro do assunto dos artefactos é mitigado através de comentários feitos pelo autor nas secções identificadas para inspecção.

O *workflow* de uma *Inspection* consiste numa fase inicial de planeamento, onde o *PM (*Project Manager) deverá eleger o moderador e fornecer as datas disponíveis para a *Inspection*. O moderador por sua vez deverá escolher a equipa e os respectivos papéis de cada colaborador, agendar a reunião, verificando a disponibilidade da equipa, e preencher o relatório *DRMM*. Os colaboradores uma vez notificados, ao consultarem o relatório *DRMM* (o relatório é criado e persiste na plataforma *Confluence*, configurada para a RLS) têm acesso a toda a informação necessária. Terminada a fase de planeamento os revisores são responsáveis por individualmente fazerem a sua preparação para a reunião de inspecção. Durante a preparação, os revisores devem inspeccionar os artefactos com o objectivo de encontrar erros, fazendo uso do material de apoio disponível (ex.: *checklists*, *templates*, *etc.*). Todos os potenciais erros encontrados devem ser registados no relatório individual *DRR* [ver anexo – E3], respeitando sempre as regras definidas pelo procedimento *Documents review*. Antes de a reunião começar o moderador é responsável por verificar se cada *DRR* foi preenchido e se os revisores fizeram

o registo das métricas. Durante a reunião deverá ser o moderador o responsável por conduzir a mesma, inspeccionando os artefactos linha a linha – página a página. O objectivo da reunião é simplesmente a identificação de erros não havendo espaço para discussão de possíveis soluções. Se não existir consenso na identificação dos erros, cabe ao moderador a última palavra. Após a reunião o autor, e/ou a equipa, têm um período de *rework* onde deveram resolver todos os erros identificados, registando a resolução dos mesmos. Na fase seguinte, *follow-up,* o moderador irá verificar em conjunto com o autor as alterações efectuadas. No entanto se durante a reunião de inspecção tiverem sido identificados mais de 3 erros críticos[9], a fase de follow-up será substituída por uma nova inspecção (preferencialmente uma *walkthrough*, por questões de *overhead*). Esta acção aplica-se a qualquer tipo de inspecção, e é feita de modo a mitigar o risco de existirem mais erros críticos ainda por descobrir, ou que novos erros tenham sido introduzidos na fase de *rework,* uma vez que erros críticos provocam alterações mais profundas, aumentando o risco dos artefactos.

A tabela 1 [ver anexo – B2, Table 1 - Review type vs document type] especifica assim os tipos de inspecções necessárias para verificar e aprovar cada tipo de artefacto. Por exemplo um *Test Plan* poderá ser aprovado apenas com uma *Ad-Hoc*, ao passo que um documento de *Requirements* terá de ser aprovado no mínimo com uma *Walkthrough*. Estas decisões foram tomadas com base no risco, estabilidade e importância de cada tipo de artefacto, e com base na experiência dos autores responsáveis pelo desenvolvimento de cada tipo de artefacto.

O risco associado aos artefactos é portanto um factor que poderá a determinar por si só o tipo de inspecção adequado. A tabela 2 [ver anexo – B2, Table 2 - Review type decision factors] tem como objectivo descrever os vários factores que influenciam o risco associado, tais como o número de alterações desde da última versão (descrito anteriormente), atributos de qualidade, pressão das *deadlines*, etc. Os atributos de qualidade são talvez o factor mais óbvio, uma vez que se estivermos a falar software para a área de saúde ou aeroespacial por exemplo, é natural que sejam exigidas inspecções formais uma vez que se trata de software com altos padrões de qualidade e de tolerância zero a qualquer tipo de erros. No caso de um projecto onde exista pressão nos prazos do projecto, numa primeira análise poderá parecer difícil compreender a razão de serem exigidas inspecções formais uma vez que estas têm um maior custo, e consequentemente consumem mais tempo aos colaboradores quando estes precisam de todo o tempo disponível para desenvolver o software.

Voltando aos benefícios do processo *Verification,* um deles é a capacidade de reduzir o *rework* associado ao desenvolvimento de software. O *rework* é reduzido de uma forma simples: os erros são detectados e corrigidos nas fases iniciais do projecto onde o custo de correcção dos mesmos é mais baixo; desta forma reduz-se o risco de existirem erros por detectar, que uma vez identificados apenas nas fases finais implicam custos (tempo) elevados para serem corrigidos. Posto isto é fácil compreender a necessidade de serem exigidas inspecções formais, especialmente quando os prazos para o projecto estão sobre pressão.

Surge então a pergunta: "Qual a razão para não serem feitas sempre inspecções o mais formais possíveis (*Inspections*)?". Num cenário ideal a empresa teria apenas um projecto de cada vez, com todos os seus colaboradores alocados para esse projecto. Desta forma, seria ideal optar sempre por *Inspections,* já que toda a equipa partilhava o mesmo calendário. No entanto num cenário real a empresa possuí vários projectos a decorrer ao mesmo tempo, com os seus colaboradores distribuídos por vários projectos. Sendo uma *Inspection* constituída por revisores dependentes e independentes do projecto (é aconselhável serem, sempre que possível,

[9] Este número foi decidido em conjunto com a RLS, e poderá ser alvo de ajustes. Não havendo a possibilidade de tomar uma decisão com base em factos, a decisão recaiu na experiência de colaboradores experientes e já familiarizados com actividades de inspecção.

independentes), um agendamento de uma *Inspection* terá de verificar não só a disponibilidade dos revisores no calendário do projecto em questão como também no calendário dos colaboradores independentes. Por esta razão, e pela importância de V&V*,* existem várias empresas que optam por criar departamentos internos e independentes de V&V (IV&V), como NASA ou a empresa portuguesa Critical Software (primeira empresa portuguesa a atingir o nível 5 do modelo CMMI).

Na tabela 3 [ver anexo B2 - Table 3 - Review type decision objectives] foram especificados os objectivos que cada tipo de inspecção pretende cumprir, onde se destacam os objectivos:

- *Ad-Hoc*
    - Procurar e desenvolver soluções.
    - Minimizar custos de inspecção.
- *Inspection*
    - Medir a qualidade dos artefactos
    - Validar a conformidade dos artefactos com as especificações e *standards*.

A qualidade providenciada pelas *Inspections* é maioritariamente assegurada pelo facto dos artefactos serem complemente revistos em duas fases, preparação e reunião, e também por ser exigido o uso de *checklists* durante a fase de preparação de cada revisor, ao rever os respectivos artefactos. Embora seja mais comum utilizar as checklists em *Inspections*, nada impede sejam utilizadas em qualquer outro tipo de inspecção. Aliás, é aconselhável que no futuro cada colaborador da RLS construa a sua própria *checklist*, com os seus erros mais comuns e a partilhe com a equipa e a empresa, para ser integrada na lista de referência.

A figura 1 [Review types activities] foi criada com o objectivo de sumarizar a informação essencial em formato visual, de modo a que seja possível aos colaboradores da RLS compreenderem todo o *workflow* dos 3 tipos de inspecções apenas recorrendo à visualização de uma imagem. Através da figura (diagrama) é assim possível conhecer quais as fases de cada tipo de inspecção, e as respectivas tarefas de cada *role* (*PM, Author, Reviewer (s), Moderator*) nas diferentes fases e tipos de inspecção. É importante referir ainda que o *workflow* das inspecções foi integrado com a plataforma *Confluence* através do processo de gestão documental [ver anexo – A3] suportado pelo *plugin ad-hoc workflows*. Isto significa que os artefactos persistentes no *Confluence* uma vez sujeitos a inspecções possuem um ciclo de vida que permite definir os seus estados (*Draft, Ready, Reviewed e Published)* em paralelo com as fases da respectiva inspecção (*Planning, Preparation, Meeting, Rework, Follow-up)*. Desta forma foi possível oferecer a capacidade de transparência ao processo *Verification*. Tomando como exemplo: se um artefacto está no estado *Ready*, significa que está pronto a ser inspeccionado. Se o estado entretanto mudar para *Reviewed*, significa que o artefacto foi inspeccionado e está agora à espera de ser aprovado. Se for aprovado, irá mudar para o estado *Published,* caso contrário voltará para o estado *Ready*. Assim, qualquer colaborador (se possuir os privilégios correctos, definidos no *Confluence*) poderá saber de imediato o estado em que um artefacto se encontra, mesmo que não participe nas inspecções.

A figura 1 [Figure 1 - Review types activities] é complementada com a secção 3. *Roles and Activities*, que de forma minuciosa descreve por cada fase as tarefas e responsabilidades de cada *role* de acordo com os 3 tipos de inspecções.

Na secção 4. Issues Classification Criteria foi especificada a classificação dos erros encontrados consoante a sua da gravidade, e os diferentes estados possíveis para cada erro. Os erros encontrados são então classificados como:

- *Low* (baixo) – O erro representa uma inconformidade com os *standards* ou formatos especificados, ou possuí erros estéticos ou ortográficos. Este tipo de erros aplica-se tanto ao código como à documentação.

- *Medium* (médio) –

    - No código este tipo de erros embora não provoquem em falhas causam resultados incorrectos, inesperados, incompletos ou abaixo do esperado *(*ex.: *low performance).*

    - Na documentação são erros que podem provocar mal-entendidos, ou que contêm falta de informação ou informação inadequada, ambiguidade ou ainda informação obsoleta.

- *Critical* (Crítico) –

    - No código são classificados como erros críticos quando provocam falhas no sistema, impedindo o seu funcionamento, ou afectam vários componentes.

    - Na documentação significam que a informação está incorrecta.

- Acrescenta-se ainda que a importância do conteúdo onde o erro foi encontrado, deverá ser tida em conta na atribuição da gravidade do erro encontrado. Os colaboradores devem assim utilizar esta classificação sempre que identificarem um erro, quer seja durante uma actividade de inspecção ou em actividades do dia-a-dia. Actualmente na RLS no que diz respeito a inspecções de documentação os erros são identificados nos próprios artefactos (através do *plugin Kwik)* e registados no relatório *DRMM,* conforme os campos definidos. Nesses campos consta também o estado de cada erro:

- *Open* – Potencial erro identificado, encontrando-se por aprovar. Se não for aprovado será descartado e não ficará sujeito a *rework*. No entanto é válido para a métrica de número de erros encontrados pelo revisor.

- *Approved* – Erro identificado e aprovado. É responsabilidade do moderador dar como aprovado potenciais erros, identificados pelos revisores.

- *In progress* – Erro a ser corrigido. É especialmente útil quando existe mais do que um colaborador responsável pelo *rework* no artefacto.

- *Not fixed* – Erro não resolvido. Deverá ser acompanhado por uma explicação (ex.: erro já não se aplica, falta de informação, não se verificou o erro, etc.).

- *Fixed* – Erro corrigido. Deverá ser verificado para que seja dado como fechado.

- *Closed* – Erro verificado e fechado.

- *Reopened* – Erro reaberto.

No que diz respeito as inspecções sobre artefactos de código os erros são registados directamente na plataforma JIRA e apenas mapeados no relatório *DRMM*. Os erros (introduzidos como *issues* no JIRA) mantêm os parâmetros (*severity type, status, description, location* e *origin)* definidos, e são controlados pelo *workflow* especificado no JIRA. Para que seja possível os erros (*issues*) manterem os mesmos parâmetros do relatório *DRMM* é necessário adicionar os campos em falta em "*issues types*" no JIRA. O *workflow* do JIRA também é configurado de acordo com os estados especificados no procedimento, acima descritos. A integração do processo *Verification* com o JIRA possibilita assim:

- Transparência – os erros uma vez inseridos são de imediato visualizados pela equipa de desenvolvimento (especialmente útil se a inspecções tiver sido desempenhada por colaboradores independentes ao projecto).
- *Tracking* – durante todo o tempo de vida do erro é possível registar de forma automática informação acerca dos estados, responsáveis pelas correcções, comentários e anexos e tempo até resolução do erro.
- Medir para avaliar – Do *tracking* é possível extrair informação como a velocidade de resolução dos erros, tempo até resolução, variações no número de erros encontrados, etc.

Ainda com o objectivo de tirar o máximo proveito do JIRA, integrando-o com o processo de *Verification* e com visão sobre o futuro da RLS onde será possível de forma automatizada cada colaborador registar o seu tempo gasto em tarefas (time tracking), as reuniões de cada inspecção (*Walkthrough e Inspection*) são inseridas no JIRA como *issues*. Se a reunião for do tipo *Inspection* é criada uma *sub-task* por cada revisor no *issue* da reunião para representar o tempo gasto na fase de preparação. Isto permite que os revisores sejam notificados para a necessidade de medir o tempo gasto na fase de preparação, e que o moderador possa verificar e facilmente recolher os tempos antes do início da reunião.

O procedimento *Documents Review* termina com a secção 5. *Documents Information*, que descreve os acrónimos e definições usados no documento, documentos aplicáveis e tabela de versões e revisões do procedimento.

6.5.1 Resultados Inspecções

Dos benefícios já mencionados relacionados com a redução de *rework*, aumento da qualidade, aumento da produtividade, etc., em que todos estes relacionam-se directamente com a competitividade da empresa, existem ainda outros benefícios que o processo oferece, que embora não possam ser medidos financeiramente, não deixam de contribuir significativamente para o sucesso da empresa. Desses benefícios podem-se destacar [25,26]:

- Desenvolvimento de *team work* e *team spirit*. O código passa a "pertencer" à equipa, uma vez que toda a equipa contribui para melhorar as partes desenvolvidas por cada colaborador.
- Promove a partilha de conhecimentos. É especialmente importante quando inesperadamente um colaborador tem de abandonar o projecto.
- Constitui uma forma de treino para os colaboradores mais recentes, uma vez que ao participarem nas inspecções irão rapidamente tornar-se familiares com os *standards* da empresa, com a cultura e métodos de trabalho; a possibilidade dos colaborares mais experientes fazerem aos mais recentes a introdução às tecnologias e técnicas utilizadas nos projectos.

- Ao serem identificadas as causas dos erros, é possível identificar melhorias no processo de desenvolvimento de software com o objectivo de prevenir que os mesmos erros possam acontecer no futuro.

O próximo capítulo descreve toda a especificação envolvida na definição do processo *Validation* e respectivos procedimentos.

Capítulo 7
7.1 Validation

O processo de *Validation* [ver anexo – C1] engloba as actividades de software *testing* da RLS, desde do planeamento à execução e revisão das mesmas. O processo é constituído por 4 procedimentos:

- *Test Planning* [ver anexo – C2]
- *Test Design and Implementation* [ver anexo – C3]**.**
- *Test Execution* [ver anexo – C1]**.**
- *Test Review* [ver anexo – C8].

Estes procedimentos têm como objectivo detalhar cada actividade do processo, uma vez que os objectivos, critérios de entrada e saída, *inputs* e *outputs* variam de actividade para actividade.

Por sua vez o procedimento de *Test Execution* possui 4 (quatro) níveis diferentes de teste:

- Unit Testing [ver anexo – C4].
- Integration Testing [ver anexo – C5].
- System Testing [ver anexo – C6].
- Acceptance Testing [ver anexo – C7].

O processo de *Validation*, respeitando a estrutura definida para documentação de processos, é iniciado com a informação do próprio documento, tal como no processo de *Verification*. De seguida na secção 2 é feita a introdução ao processo onde se destaca o propósito do processo em avaliar as capacidades e atributos do software para atingir os requisitos, especificações e necessidades dos *stakeholders*, e ainda aumentar a confiança na equipa na qualidade actual do software. Ainda na secção 2.2 foram definidos os objectivos do processo onde é reforçado uma vez mais o objectivo de reduzir o *rework* e custos associados ao desenvolvimento de software, através da detecção de erros nas fases iniciais. Refere-se que uma vez mais os *Outcomes* da secção 2.3 foram definidos pela RLS.

Na secção 3 *Process Overview* foi criada a figura 1 que pretende representar todo o ciclo de vida do processo. No entanto como a RLS não está sujeita a uma metodologia pré-determinada, mas sim a escolha de uma que melhor se enquadre com o projecto a desenvolver, o processo de *Validation* foi especificado para ser implementado tanto em metodologias ágeis (ex.: *XP, SCRUM*) como em metodologias de uma iteração apenas (ex.: *waterfall).* Essa adaptação é representada na figura 1 – *Validation process map* pela seta tracejada a vermelho *"Project iterations",* uma vez que num projecto SCRUM por exemplo, devido ao número de *sprints* (*"project iterations"*) os procedimentos serão executados (não necessariamente todos) várias vezes, até que o processo termine com o projecto. A figura mostra também todos os *inputs* e *outputs* dos procedimentos, as oportunidades de melhorias contínuas e a relação do processo *Validation* com os restantes processos da RLS.

A secção 3.1 *start* e *end events* do processo são definidos como o início da fase de análise de requisitos do projecto e como a conclusão do contracto do projecto respectivamente. Idealmente este processo deverá decorrer em paralelo com todo o processo de desenvolvimento, uma vez que à medida que são definidas as especificações dos requisitos, *design* do sistema, arquitectura, etc. deveram ser especificados também os casos de teste para validar as respectivas especificações (ver figura 3 – V&V overview). As condições de entrada e saída *(3.2 Entry and Exit Conditions),* sendo meios de validação dos eventos de início e fim do

processo, foram definidas para garantir que o processo só será iniciado depois de 2 documentos essenciais *(Business Requirements e Project Plan)* para o procedimento inicial *Test Planning* tenham sido aprovados (verificados pelo processo de *Verification*), e também para que o processo possa terminar depois do *Test Plan* ter sido cumprido, os erros encontrados terem sido reportados (no *JIRA*) e por fim que o relatório [ver anexo – E5] tenha sido aprovado.

Na secção 3.3 de *inputs* e *outputs* do processo destaca-se o conjunto de documentos: *Requirements specifications* e Software *specifications*; que em paralelo serão verificados pelo processo de *Verification* para serem utilizados como *inputs* em todo o processo de *Validation*, mas o maior foco será no procedimento de *Test Design and Implementation* uma vez que é neste procedimento que os casos de teste são criados. Destaca-se ainda como *input* o *Test Plan* (*template*) que será criado no procedimento de *Test Planning*, servindo depois de input para os restantes procedimentos do processo. No que diz respeito aos *outputs* destaca-se o software testado, que é o grande objectivo do processo, mas também o relatório Software Testing Report que tem como objectivo resumir e registar num só relatório as evidências de maior importância (métricas, resultados, actividades desempenhadas, etc.) da componente de software *testing* (*Validation*) do projecto.

7.2 Testing Methods

No processo *Validation* houve ainda a necessidade de especificar a *baseline* de métodos de teste de software que serão usados na RLS. No entanto os métodos de teste são vastos e cada um diferente na sua natureza e objectivos, o que implica que a *baseline* estará sujeita a um possível *tailoring* (alterar ou adaptar procedimentos/actividades do processo), sempre que este seja definido e correctamente justificado no *Test Plan* pelo *Testing Leader*. Os métodos de teste são compostos pelas categorias:

- *Testing Types* [Validation process, 4.1 Testing types] –
    - Utilizados para definir que parte do software será testada, isto é, a parte funcional ou a estrutura do software.
- *Testing Techniques* [Validation process, 4.2 Testing techniques] –
    - Utilizadas para definir como a parte específica do software será testada através de diferentes técnicas, cálculos ou métodos.

Os tipos de teste *4.1 Testing types* foram classificados em duas categorias:

- *Black Box* – diz respeito às partes funcionais do software, onde os casos de teste são derivados das especificações de requisitos e design do sistema, e testados contra essas mesmas especificações indicando se as especificações são ou não cumpridas. Neste tipo de teste o *tester* não tem acesso ao código do software, como se o software se trata-se de uma caixa preta onde seja possível apenas inserir *inputs* e observar os *outputs* produzidos. Por esta razão o tipo de teste *Black Box* garante que o software implementa as funcionalidades especificadas, satisfazendo as necessidades dos *stakeholders*.
- White Box – diz respeito à estrutura do software em si, onde os casos de teste são derivados das especificações de implementação (*Low Level Specifications*) para testar e encontrar erros de implementação. Aqui o *tester* tem acesso total ao código do software, pelo que o objectivo é observar as operações do software durante o processamento de *inputs* e *outputs*. Assim ao testar o software através de testes do tipo *white box* é possível garantir que o software funciona de forma correcta.

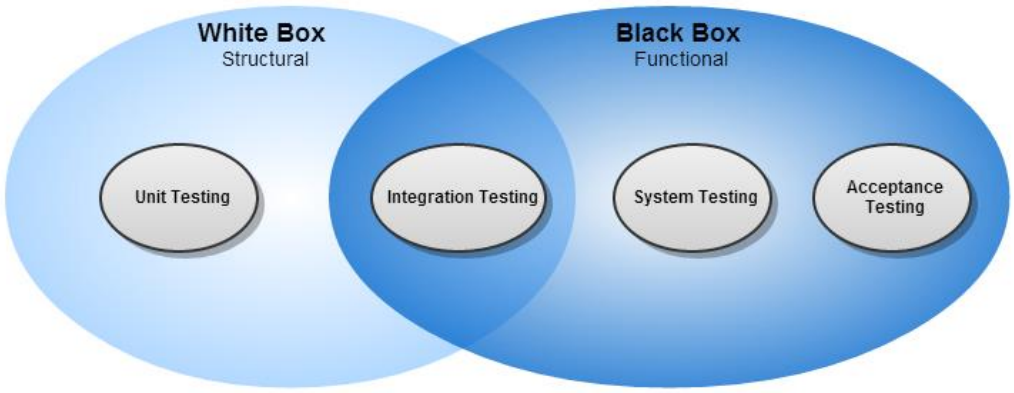


Figura 23 - White Box vs Black Box

Para que o software seja testado e validado de forma completa, é necessário que sejam aplicados ambos os tipos de teste, *White* e *Black Box*. No entanto o tipo de teste *White Box* é bem mais caro que o *Black Box* **[28]**, uma vez que são necessários colaboradores com conhecimentos técnicos de *testing, coding* e *code compreension* para a especificação e implementação correcta de casos de teste eficientes, e dado que é incomum testar exaustivamente o software, mesmo que relativamente pequeno, devido ao tempo consumido no processamento dos casos de teste *White Box*, uma vez que são necessário vários (talvez dezenas) de testes unitários para cobrir uma única unidade.

A figura 4 - *Testing types overview* define a que tipo de teste pertence os diferentes níveis de teste (Unit, Integration, System e Acceptance), onde é possível ver que *White Box* por ter como objectivo testar as especificações de implementação é testado ao nível das unidades implementadas (*Unit Testing*), e que *Black Box* por ter como objectivo testar as funcionalidades do software é testado ao nível do sistema (*System Testing)* e ao nível da fase final de aceitação (*Acceptance Testing*) do software desenvolvido. Ao nível de integração (*Integration Testing)* por ser objectivo testar as interfaces entre os componentes, foi definido que este partilha os 2 (dois) tipos de teste *White e Black Box*. Sendo posicionado entre os tipos de teste *White* e *Black Box*, o *tester* terá acesso ao código do software (*White Box)* necessário para desenvolver casos de teste que permitam testar o software ao nível das funcionalidades do sistema (*Black Box*). No entanto na literatura é comum fazer-se referência a este posicionamento entre *Black* e *White Box* como *Gray Box*, onde é definido que o *tester* tem acesso apenas a parte do código **[29, 30]**.

7.3 Testing Techniques

No que diz respeito às técnicas de teste foram especificadas no processo apenas técnicas do tipo *Black Box,* ou seja ao nível de testes do sistema e testes de aceitação. As razões que levaram a esta decisão foi o facto das técnicas *White Box* serem muito mais técnicas e portanto não estariam correctamente enquadradas no processo, uma vez que nem todos os colaboradores teriam conhecimentos técnicos para as interpretar, e por outro lado pela diversidade de técnicas existentes seria mais difícil definir uma *baseline* comum a projectos. No entanto na

secção *White Box* serão também listadas algumas das técnicas usadas, recolhidas durante o processo de análise de diferentes técnicas de teste.

7.3.1 Black Box Testing

7.3.1.1 *Regression testing*

Consiste numa selecção completa ou parcial dos casos de teste já executados, que ao serem reexecutados indicam se as alterações feitas no código causaram efeitos indesejados no comportamento do software, e se o sistema e os seus componentes continuam a cumprir as suas especificações e funcionalidades. Esta técnica deverá ser aplicada frequentemente, sempre que existam alterações significativas no código devido a correcções de bugs ou implementação de novas funcionalidades. Para que esta técnica possa ser aplicada correctamente sem causar quebras na *performance* da equipa, a execução dos casos de teste deve ser, sempre que possível, feita através de scripts que executem as *suites* de teste (conjunto de casos de teste agrupados) e que os casos de teste escolhidos possam ser executados durante o período de inactividade da equipa, ex.: durante a noite e o durante o fim-de-semana.

Esta técnica poderá também ser usada para obter indicadores de risco do *software* em desenvolvimento, ou seja, uma vez analisados os primeiros resultados dos erros encontrados durante a técnica de *Regression*, se durante as próximas iterações da técnica o Nº de erros encontrados aumentar mais do que esperado, poderá significar que as alterações actuais representam um risco maior para a performance da equipa devido a factores relacionados com a complexidade, uso de novas tecnologias ou maiores dependências entre componentes. Desta forma poderá ser possível prever esse risco e actuar conforme necessário, ex.: aumento do número de inspecções, treino da equipa, etc.

Para esta técnica foi especificado:

- Objectivos – Assegurar que as alterações não afectaram os componentes ou funcionalidades do sistema já implementadas.
- Técnica -
    - Escolher casos de teste que testem os componentes e funcionalidades do software já implementadas;
    - Escolher casos de teste que sejam focados nos componentes e funcionalidades que tenham sido alteradas recentemente.
    - Escolher casos de teste adicionais que sejam focados nos componentes e funcionalidades que tenham probabilidade de serem afectados pelas alterações.

7.3.1.2 *Functional testing*

Esta técnica foca-se em validar se o sistema desenvolvido cumpre os requisitos especificados e as suas funcionalidades através do mapeamento directo dos casos de teste para casos de uso, requisitos de negócio e respectivas restrições. Por ser uma técnica pura de *Black Box* é necessário validar o sistema e os seus processos internos através da interacção com a aplicação do sistema, identificando e definindo em primeiro lugar os *inputs* e resultados esperados, para depois executar os casos de testes e analisar os outputs produzidos. Uma vez que é exigido o correcto funcionamento do software para aplicar esta técnica, a mesma deverá ser aplicada durante o nível de testes do sistema e testes de aceitação.

Para esta técnica foi especificado:

- Objectivos – Garantir os requisitos e funcionalidades adequadas, incluindo navegação, registo de dados, processamento, etc.
- Técnica – Executar cada caso de teste usando inputs válidos e inválidos para validar:
    - Os resultados esperados ocorrem quando é utilizado inputs válidos.
    - Mensagens de erro ou avisos são mostradas quando é utilizado inputs inválidos.

7.3.1.3 *User interface testing*

O teste da UI (*User Interface*) valida a interacção do utilizador com o software. O seu objectivo é assegurar que a UI oferece ao utilizar o acesso e a navegação apropriada pelas funcionalidades do sistema. Adicionalmente, esta técnica assegura ainda que os objectos dentro da UI funcionam como esperado e em conformidade com os standards definidos pela RLS.

Para esta técnica foi especificado:

- Objectivos –
    - Validar se navegação pelo software reflecte correctamente as funcionalidades e requisitos especificados, incluindo o uso de métodos de *input (*rato, teclado, etc.).
    - Validar as janelas e os seus objectos e as suas características, como os menus, o tamanho, a posição e as suas conformidades com os standards.
- Técnica – Criar casos de teste para cada janela para validar a navegação e o estado dos objectos de cada janela da aplicação.

7.3.1.4 *Performance testing*

Esta técnica, tal como o nome indica, consiste em testes de performance onde os tempos de resposta, tempos de execução, número de transacções por segundo/minuto e outros requisitos baseados em tempo são medidos e avaliados, com o objectivo de validar se os requisitos de performance são cumpridos. A técnica é implementada e executada para que seja possível ajustar os comportamentos de desempenho do software em função de condições tais como a carga de trabalho ou as configurações de hardware do sistema.

Para esta técnica foi especificado:

- Objectivos – Validar a performance com os requisitos especificados, nas condições de carga de trabalho esperadas.
- Técnica
    - Simular carga de trabalho de vários clientes através de utilizadores virtuais.
    - Aumentar o número de pedidos nos casos de teste, e criar scripts para aumentar o número de iterações de cada pedido.
    - Utilizar múltiplos clientes físicos onde cada cliente corra os scripts de teste para provocar um aumento da carga no sistema.

7.3.1.5 *Stress testing*

Esta técnica é constituída também por testes de performance, onde o software é sujeito a diferentes cargas de trabalho para medir e avaliar os comportamentos de performance e

capacidade do software em continuar a funcionar correctamente sob diferentes cargas de trabalho. O objectivo principal é determinar e assegurar que o sistema funciona correctamente quando a carga de trabalho é superior à sua capacidade prevista.

Para esta técnica foi especificado:

- Objectivos – Validar o comportamento do software sob diferentes condições de carga de trabalho extremas.
- Técnica
    - Utilizar os casos de teste desenvolvidos para *Performance Testing*.
    - Utilização de múltiplos clientes, onde sejam executados os mesmos casos de teste ou casos de testes adicionais, para simular o "pior cenário" possível de carga durante períodos de tempo prolongados.

7.3.1.6 *Security testing*

Esta técnica foca-se em dois níveis de segurança: Aplicação e Sistema. A segurança ao nível da Aplicação assegura que, baseado nos critérios de segurança definidos, os utilizadores são restritos de funcionalidades ou acções, ou os dados disponíveis de para acesso são limitados. Ou seja, todos os utilizadores poderão inserir dados e criar novas contas, mas apenas os utilizadores com privilégios de gestor poderão apaga-los. Ao nível de segurança do Sistema é assegurado que apenas os utilizadores com privilégios de acesso ao sistema serão capazes de aceder às aplicações e apenas através de ligações apropriadas (ex.: intranet).

Para esta técnica foi definido:

- Objectivos –
    - Segurança ao nível da Aplicação – Validar se um utilizador tem acesso apenas às funções e dados permitidos pelo seu tipo de utilizar.
    - Segurança ao nível do Sistema – Validar que apenas os utilizadores com acesso ao sistema e aplicações, pode aceder às mesmas.
- Técnica –
    - Criar casos de teste para cada tipo de utilizar e verificar cada permissão criando operações específicas para cada tipo de utilizador.
    - Modificar o tipo de utilizador e executar novamente os casos de teste, verificando se as funcionalidades ou dados adicionais foram correctamente negadas ou disponibilizadas.

7.3.1.7 *Fail-over and recovery testing*

Esta técnica assegura que o Sistema tem a capacidade de recuperar com sucesso de situações causadas por falhas de hardware, software ou de rede, sem que seja posto em causa a perda ou a integridade dos dados. A técnica de *fail-over* assegura que nos Sistemas que têm de estar sempre em execução, quando uma condição de *fail-over* acontece, o Sistema de backup (ex.: Sistemas Distribuídos) assume o comando do Sistema primário, sem que exista perdas de dados ou operações. A técnica de *recovery* é baseada em testes onde o Sistema é exposto a condições extremas que causem falhas de *I/O (Input and Output)* ou entradas inválidas na base de dados, para depois o Sistema ser monitorizado e inspeccionado com o objectivo de validar se o sistema e os dados foram recuperados correctamente.

Para esta técnica foi definido:

- Objectivos – Verificar que os processos de recuperação (manuais ou automáticos) restauram propriamente a base de dados, aplicações, dados e Sistema para um estado desejado e conhecido (como "saudável").
- Técnica – Os casos de teste criados para a técnica de *Functional testing* devem ser utilizados para criar um conjunto de operações. As seguintes situações devem então ser criadas ou simuladas, individualmente:
    - Desligar a energia do cliente.
    - Desligar a energia do servidor.
    - Interromper a rede, simulando falhas na comunicação (ex.: routers, cabos de rede, etc).
    - Abortar, corromper ou terminar prematuramente operações da base de dados.
    - Assim que as condições anteriores tiverem sido simuladas, os procedimentos de recuperação devem ser iniciados.

7.3.2 White Box Testing

7.3.2.1 *Control Flow*

A técnica de *Control Flow* é a mais comum dentro do tipo *White Box* pela possibilidade de ser aplicada a praticamente todo o software, sendo altamente eficaz **[28,31]**. Estima-se que a técnica de *Control Flow* consegue identificar cerca de 50% dos bugs durante a fase de *Unit Testing*. Esta técnica possui vários métodos de teste, que têm como objectivo testar diferentes aspectos do software, alcançando a maior de cobertura de código testado possível. A percentagem de cobertura dos testes é conhecida como *Coverage* (métrica de *code coverage*). Os métodos de cobertura de código presentes nesta técnica são:

- *Statement Coverage* – métrica que indica a percentagem de instruções que foram executadas pelos casos de teste. É também conhecida como *Line Coverage* ou *Segment Coverage*. O objectivo neste método será atingir os 100% de cobertura, embora na prática valores iguais ou superiores a 95% serão o mínimo desejado **[32]**.
- *Branch Coverage* – métrica que indica a percentagem de condições de decisão (ex.: IF, FOR, WHILE.) no software que foram avaliadas pelos casos de teste. É também conhecida como *Decision Coverage*. Neste método as condições de decisão são testadas de ambas as formas, positivas e falsas (*TRUE and FALSE*), para validar todas as decisões possíveis no código, verificando que não existem "ramos" que levam a comportamentos anormais da aplicação. Para uma boa cobertura são esperados valores iguais ou superiores a 90% **[32]**.
- *Condition Coverage* – métrica utilizada para medir a percentagem de expressõe*s Boolean* no código que foram avaliadas pelos casos de teste de ambas as formas, positivas e falsas.
- *Path Coverage* – métrica que indica a percentagem de caminhos possíveis em cada função do código, que foram percorridos pelos casos de teste.

7.3.2.1.1 *Loop testing*

É um tipo de técnica que se foca exclusivamente em validar a construção dos loops (ciclos). Os ciclos são simples de testar, desde que não existam dependências no ciclo, no código contido ou entre ciclos. Existem 4 classes de ciclos:

- *Simple Loop*
- *Nested Loop*
- *Concatenated Loop*

7.3.2.1.2 *Simple Loop*

Trata-se de um ciclo simples. Para testar *simple loops* pode-se usar o seguinte conjunto de testes, onde *n* representa o número máximo de iterações possíveis do ciclo:

1. Ignorar o ciclo.
2. Iterar apenas uma vez no ciclo.
3. Iterar duas vezes no ciclo.
4. Iterar *m* vezes no ciclo onde $n > m$.
5. Iterar *n-1, n, n+1* vezes no ciclo.

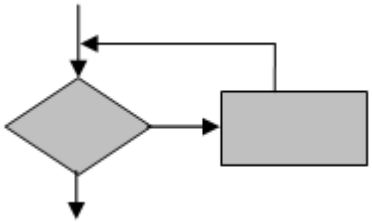


Figura 23 - Ciclo Simples

7.3.2.1.3 *Nested Loop*

Nested loop é quando existe um ciclo dentro de outro ciclo. O número de possíveis testes aumenta geometricamente à medida que são acrescentados ciclos simples ao ciclo principal.

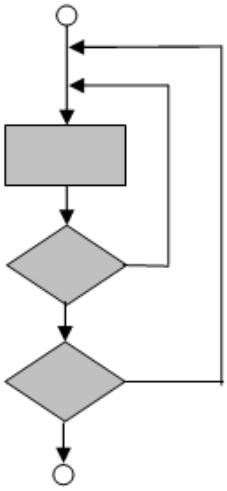


Figura 24 - Nested Loop

7.3.2.1.4 *Concatenated Loop*

Se 2 (dois) ciclos forem independentes entre si podem ser testados com a mesma abordagem dos *Simple Loop*. No entanto se o contador do primeiro ciclo for utilizado como valor inicial no segundo ciclo, então os ciclos não são independentes.

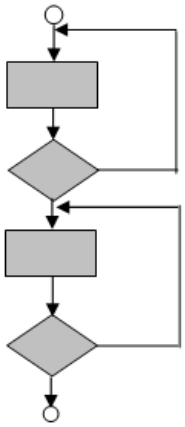


Figura 25 - Concatenated Loop

7.3.2.2 *Ciclomatic Complexity*

Não se trata de uma técnica mas sim de uma métrica utilizada para medir a complexidade do software, medindo o número de caminhos independentes no código. Um caminho independente é definido como um caminho que tem pelo menos uma aresta que não foi percorrida anteriormente por nenhum outro caminho. Para ajudar a representar as várias estruturas de controlo é utilizada a técnica de *Flow Graph* ***[34]***:

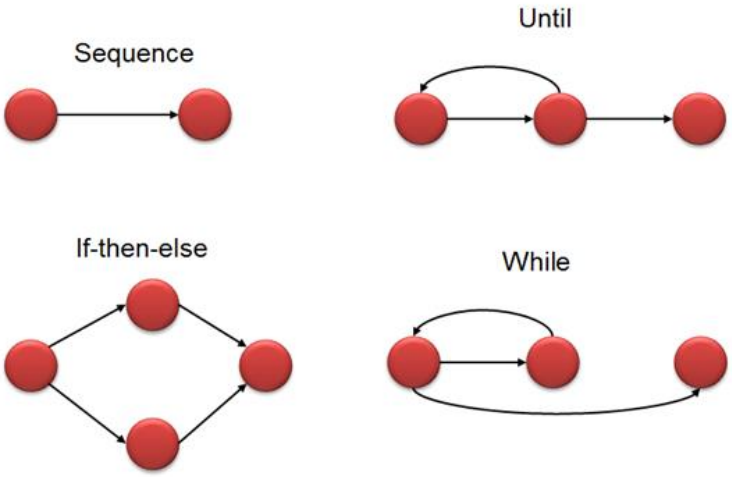


Figura 26 - Flow graph

Onde um nó representa uma condição de decisão (ex.: IF) e as arestas representam as ligações entre os nós. Apesar de existir várias formas de calcular a Complexidade Ciclomática, a forma mais fácil é somar o número de instruções de decisão (ex.: IF, WHILE, DO, etc.) e adicionar 1 ao resultado **[33]**. Ou seja: V (G) = P + 1, onde:

- V (G) – Complexidade Ciclomática.
- P – Nó de decisão.

Tomando como exemplo o seguinte pedaço de código:

```
1  IF A = 354
2      THEN IF B > C
3          THEN A = B
4          ELSE A = C
5      ENDIF
6  ENDIF
7  Print
8
```

Figura 27 - Exemplo código para cálculo V (G)

Onde o respectivo *Flow Graph* seria o da figura seguinte:

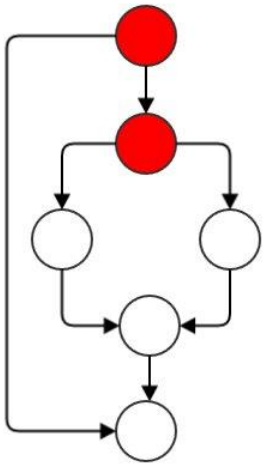


Figura 28 - Flow Graph

Na figura 6 podemos observar que existem 2 nós de decisão, representados a vermelho, uma vez que possuem duas saídas possíveis. Assim pela fórmula anterior a Complexidade Ciclomática deste pedaço de código é igual a 2 + 1 = 3. Ou seja, seriam necessários 3 casos de teste para obter uma *Path Coverage* de 100%.

Uma das maiores vantagens em calcular a complexidade do código é a capacidade de utilizar esta métrica como forma de avaliar o risco associado ao código. Atendendo à tabela abaixo **[35]**:

Cyclomatic Complexity	**Risk Evaluation**
1 – 10	Código simples – pouco risco.

11 – 20	Código um pouco complexo – risco moderado.
21 – 50	Código complexo – risco elevado.
51 =<	Código não testável – risco demasiado elevado.

Verificamos que o código de exemplo (figura 27) é simples e portanto de pouco risco, o que significa uma probabilidade menor de potenciais erros no código. No entanto é difícil definir o limite máximo para a complexidade do código, por norma são esperados valores máximos de 10. Apesar de existir projectos complexos onde são aceites valores máximos entre 10 e 15, é importante referir que uma vez aceites estes limites de complexidade, é esperado que o projecto possua uma equipa de colaboradores experientes e uma forte componente de V&V **[36]**. Para tornar esta métrica ainda mais completa, e para que seja possível obter uma estimativa mais precisa do risco associado, é usual combinar a complexidade com o número de linhas do código (LOC), uma vez que a métrica *LOC* está também associada ao número de erros encontrados **[36]**. Estudos mostram que o código com alta complexidade e de grande tamanho (LOC) possuem um risco maior. O mesmo acontece com o código pequeno (LOC) mas com alta complexidade uma vez que tende a ser código bastante conciso, e portanto difícil de alterar e modificar **[36]**. Deve-se portanto encontrar uma relação "saudável" entre a complexidade e o tamanho do código desenvolvido.

7.3.2.3 *Mutation testing*

Mutation testing consiste em alterar certas instruções, mantendo a sintaxe correcta, no código para depois verificar se os (mesmo) casos de testes são capazes de encontrar os erros introduzidos. As alterações deverão ser pequenas para que os objectivos globais do código não sejam afectados. Embora pertença ao tipo de testes *White Box*, não se trata de uma técnica de teste mas sim uma estratégia para medir e avaliar a qualidade dos casos de teste desenvolvidos, que devem ser suficientemente robustos para detectar os erros no código mutado. Se os casos de teste não detectarem os erros no código mutado, pode significar que os casos de teste estão incompletos ou incorrectos.

No entanto esta estratégia tem custos (tempo) bastante elevados, uma vez que não só é necessário gerar as mutações, como também correr novamente os casos de teste para as respectivas mutações. Por norma este tipo de estratégia é apenas praticável com recurso a ferramentas automatizadas de teste. Apesar dos custos, esta estratégia é utilizada de forma bastante vantajosa para atingir objectivos como altos índices de cobertura do código, qualidade dos casos de teste e ainda determinar a probabilidade de existir erros por identificar no código.

7.4 Validation Activities

Nesta secção 5 - *Validation Actitivies* foram descritos de forma detalhada os 4 procedimentos do processo, assim como as responsabilidades dos diferentes *roles* que participam nas actividades de teste. A figura 5 [ver anexo – C1] pretende assim ilustrar num único diagrama a relação entre os procedimentos e as tarefas necessárias de cada *role,* sendo possível observar também o *workflow* desejado entre os 4 procedimentos. As tarefas e responsabilidades de cada *role* são então descritas em detalhe na secção seguinte, 5.1. Roles and activities. Voltando um pouco atrás e seguindo o *workflow* da figura 5 observámos que o processo é iniciado com o procedimento de *Test Planning* onde a responsabilidade do *PM (Project Manager)* será reservar tempo suficiente para as actividades de *software testing* (as actividades são constituídas pelos quatro procedimentos do processo) dentro do calendário do projecto definido no documento

de *project plan*. Uma vez especificado o tempo disponível para as actividades de *software testing*, o *Testing Leader* é responsável por criar o documento de *Test Plan* que incluí a especificação de ambientes de teste, ferramentas, estratégias e níveis de teste adequadas ao projecto, etc. Ainda na fase de *Test Planning* os *testers* devem iniciar assim que possível a configuração de máquinas, ferramentas de teste, cenários de teste, etc.

A fase seguinte diz respeito ao procedimento de *Test Design and Implementation* onde o objectivo é especificar e desenvolver os casos de teste. Os responsáveis por estas actividades são os *testers,* e deverão iniciar as mesmas assim que existam especificações (requisitos, design, arquitectura, etc.) aprovadas do projecto a desenvolver.

O procedimento seguinte, de execução, é dividido em 4 níveis de teste: *Unit testing, Integration testing, System testing e Acceptance testing*; Em cada nível de teste o objectivo é executar, avaliar e registar os resultados dos casos de teste, no entanto os níveis diferem nas suas naturezas e objectivos. No entanto é possível existir situações onde níveis de testes diferentes decorram em simultâneo, ex.: *Unit e Integration; Integration e System*. Nesta situação não se espera que os níveis sejam iniciados em simultâneo, mas possa ocorrer um período em que o nível seguinte seja iniciado enquanto o nível anterior ainda está a terminar. Ainda na fase de execução foi tomada uma decisão importante relacionada com o objectivo de reduzir o *overhead* associado às actividades de *software testing*, em concreto no registo dos erros encontrados. Assim sendo foi definido que quando um caso de teste falha (um caso de teste falha se encontrar um erro, sendo portanto um caso de teste com sucesso) não é necessário abrir um *issue* (os erros encontrados são registados na plataforma *JIRA* com o nome de *Issues*) se o erro for resolvido e testado novamente com sucesso. Se o erro persistir ou não for resolvido de imediato, então é necessário registar o erro. No entanto é importante referir que todos os casos de teste falhados (métrica de casos de teste com sucesso) são obrigatoriamente registados.

A última fase diz respeito ao procedimento de *Test Review* onde o objectivo é fazer o registo dos resultados das actividades de execução. Cada *tester* é responsável pelo registo das suas actividades. A responsabilidade do *Testing Leader* nesta fase é verificar que todas as actividades de teste foram executadas de acordo com os critérios especificados no *Test Plan*. O *Testing Leader* é ainda responsável por preencher o relatório Software Testing Report template [ver anexo – E5] que sumariza as actividades e os resultados da componente *software testing* do projecto. Apesar de ser o *Testing Leader* o responsável pelo relatório, é aconselhável que exista uma reunião de equipa com o objectivo de recolher o feedback do que correu bem, o que correu menos bem, o que poderá ser alterado no futuro, etc.

7.5 Procedimentos

7.5.1 Test planning

Este procedimento [ver anexo – C2] é extremamente importante para o sucesso das restantes actividades de *software testing*, uma vez que é essencial a existência de um plano correctamente adequado ao projecto. Os objectivos do procedimento Test Planning consistem na especificação do *Test Plan* e da *RTM (Requirements Traceability Matrix*). O *Test Plan* é um documento de planeamento deverá descrever os seguintes aspectos:

- *Scope* -
    - *Test items* – Os itens que serão testados do software e hardware, incluindo requisitos funcionais e não funcionais.
    - *Features to be tested* – As funcionalidades especificadas que serão testadas do ponto de vista do utilizador (requisitos funcionais).
    - *Exclusions* – Os itens que não serão sujeitos a testes.

- *Testing Strategy* –
    - *Testings types.*
    - *Testing techniques.*
    - *Testing levels.*
    - Critérios para terminar os testes.
    - Ferramentas de teste e tecnologias.
    - Estratégia de *Regression testing,* se utilizada.
    - *Estratégia de Mutation testing,* se utilizada.
    - Necessidade de recolher métricas específicas para o projecto (*MTBF – Mean Time Between Failure*, *Availability,* etc.).
- *Quality* –
    - Critérios de aprovação/reprovação (ex.: *Code Coverage*) dos itens sob teste, que deverão ser definidos por cada nível de teste.
    - *Test deliverables* – Documentos e outros artefactos que serão produzidos (ex.: simulações, relatórios, etc.).
- *Time* –
    - Calendário – baseado em estimativas realistas e válidas, incluindo o tempo e esforço estimado, e quando deveram ser desempenhadas as tarefas e actividades do processo.
- *Resources* –
    - Orçamento – Especificar quaisquer restrições associados ao orçamento disponível (tempo, dinheiro, *staff* disponível, etc.).
    - Ambientes de teste – Os recursos necessários para testarem o software e o hardware (hardware específico, versões específicas do software utilizado, simuladores, ferramentas de teste específicas, etc.).
    - Responsabilidades – especificar quem será responsável por definir o plano de riscos, formação da equipa, resolver conflitos no calendário, assegurar a aquisição dos recursos necessários, etc.
- *Staff* – Colaboradores e respectivas *skills* necessários para cumprir o *Test Plan,* com sucesso. Incluí a formação necessária, necessidade *roles* específicos *(Quality Control/Assurance Engineer, Test Designer, etc.*), etc.
- *Risks* –
    - Critérios de suspensão e requisitos de recomeço – definem sob que circunstâncias as actividades de teste devem parar e recomeçar.
    - Plano de risco e mitigação – onde são definidos os riscos e as suas probabilidades, impactos e acções de mitigação. Os riscos estão relacionados com a complexidade do software, instabilidade dos requisitos e designs, ferramentas de teste inadequadas, etc.
- *Project Milestones* –
    - *Milestones* – devem ser definidas para que seja possível acompanhar o estado e esforço do projecto.

O *Test Plan* é assim essencial para que todas as actividades de *software testing* sejam definidas e planeadas correctamente, de acordo com o projecto a decorrer. Para que o *Test Plan* seja especificado correctamente a informação necessária deverá ser capturada directamente dos *stakeholders, project plan* e outros documentos relevantes para o projecto. O outro objectivo do

procedimento é a especificação da RTM, que tem como objectivo mapear os requisitos especificados com os casos de teste desenvolvidos. Quando se específica uma RTM existem 3 abordagens possíveis:

- *Forward traceability* – Os requisitos são mapeados para os casos de teste.
- *Backward traceability* – Os casos de teste são mapeados para os requisitos.
- *Bi-directional traceability* – O mapeamento é entre requisitos e casos de teste é bidireccional.

Para a RLS foi especificado um *template* RTM bidireccional, tirando vantagem das capacidades de interoperabilidade do *Confluence* e *JIRA*. Ou seja, numa RTM facilmente será possível clicar no ID do requisito e ser encaminhado até ao respectivo documento ou clicar no ID do caso de teste e ser encaminho para a sua localização e respectiva descrição no JIRA. A RTM oferece uma visão ampla da relação entre as especificações e a validação das mesmas, e ainda a capacidade de assegura que:

- O cliente sabe que o software está a ser desenvolvido conforme os requisitos.
- Todos os requisitos estão abrangidos pelos casos de teste.
- Os programadores não estão a desenvolver funcionalidades que ninguém pediu.
- É mais fácil identificar as funcionalidades em falta.
- É mais fácil identificar os casos de teste que irão precisar de ser actualizados, se existirem pedidos de alterações nas especificações.

Posto isto, a *RTM* [ver anexo – E4] deverá conter:

- ID do Requisito/Caso de uso.
- Descrição dos requisitos/casos de uso.
- Todos os requisitos, dos documentos de requisitos (*BRS, YRS, SRS, etc.*), que possam ser testados.
- Mapeamento entre os requisitos e casos de teste.

Para concluir, o uso de uma RTM durante o processo de desenvolvimento de software faz parte das boas práticas descritas no modelo CMMI, nomeadamente da área de processo de Gestão de Requisitos *(Requirements Management @ CMMI Level 2)*.

7.5.2 Test design and implementation

Este procedimento [ver anexo – C3] tem como propósito a especificação e desenvolvimento dos casos de teste. Os casos de teste têm como objectivos:

- Encontrar erros (maior foco nos níveis de *Unit* e *Integration testing*).
- Demonstrar conformidade do software com os requisitos e especificações (maior foco nos níveis de *System* e *Acceptance Testing)*.

Para que seja possível especificar correctamente os casos de teste, os *testers* devem ter conhecimentos do software em desenvolvimento, especialmente se estiverem a ser especificados casos de teste ao nível de *White Box testing*, das funcionalidades a testar, especialmente no nível de *Black Box testing,* e naturalmente das técnicas de teste. Casos de teste bem especificados permitem que as actividades de teste sejam eficientes, uma vez que são criados apenas os casos de teste necessários e não mais. É importante relembrar que casos de teste redundantes implicam mais horas de computação, e pior ainda quando existe a

necessidade dos casos de teste serem executados manualmente, implicando não só o tempo de computação adicional como também consumo de *man-hours* da equipa.

Para que um caso de teste seja especificado correctamente, deverá descrever os seguintes aspectos:

- ID – para identificar de forma única o caso de teste.
- Test item ID – para identificar a unidade/módulo/aplicação sob teste.
- Test Procedure – Os detalhes (ex.: *inputs*, passos necessários, configurações, etc.) necessários para executar o teste.
- Expected output – O resultado que é esperado no *output* do código.
- Test date – Data de execução do teste.
- Result – O resultado do teste (*Pass* ou *Fail*). Se o resultado for *Fail* significa que não foi observado o resultado esperado, e portanto poderá indicar a existência de um erro no código.
- Problem description – No caso de o teste falhar (*Fail*), o *tester* deve fazer a descrição do problema e se possível sugerir a uma solução.
- Error Type – O tipo de erro encontrado (*Low*, *Medium*, *Critical*).
- Phase Detected – A fase na qual o erro foi detectado (Unit, Integration, System, Acceptance).
- Phase Introduced – A fase na qual o erro foi produzido (Requeriments, Design, Implementation, etc.), se possível de identificar.
- Defect Status – O estado do erro (Open, Approved, In progress, Not fixed, Fixed, Closed, Reopened).
- Verified By – A pessoa responsável por verificar a correcção do erro.
- Verification Date – A data de verificação da correcção.

Os casos de teste deverão ser introduzidos no JIRA como *Issues*. Uma vez que é possível criar um *Issue* do tipo "caso de teste" e configurar com os campos necessários. As vantagens de ter os casos de teste no *JIRA* são várias, desde da capacidade de traceability, registado de actividade de forma automática (ex.: Verified By, Verification Date, Defect Status) até à capacidade de apresentar *dashboards*, em tempo real e de forma automatizada, com as fases mais eficazes na detecção de erros (*Phase Detected*), por exemplo.

7.5.3 Test Execution

A actividade de *Test Execution* é dividida em 4 procedimentos, que dizem respeito aos 4 níveis de teste definidos no processo: *Unit Testing, Integration Testing, System Testing e Acceptance Testing*. Durante todo o processo cada nível de teste poderá ser executado uma ou várias vezes, ou até não executado de todo. A execução de cada nível de teste dependerá numa primeira instância do tipo de projecto e metodologia escolhida, e por último na estratégia de teste escolhida pelo *Testing Leader*, que será especificada no *Test Plan*. Cada procedimento tem como objectivo definir os diferentes aspectos de cada nível de teste. Os aspectos definidos são:

- Propósito.
- Audiência.
- Objectivos.
- Eventos de início e fim.
- Critérios de entrada e saída.

- *Inputs* e *Outputs*.

7.5.3.1 Unit Testing

Unit Testing [ver anexo – C4] é o nível mais baixo de teste, onde são testadas unidades (*Units*), que são as peças mais pequenas do software. Uma unidade poderá corresponder a um método ou a uma classe. *Unit Testing* tem como propósito identificar erros nas unidades e validar se o *design* da unidade foi correctamente implementado. Este nível de teste é especialmente importante por várias razões:

1. Estudos relevam que a maioria dos erros de código é detectada na fase de Unit Testing **[28, 31]**
2. Os erros identificados nesta fase são "baratos" e fáceis de corrigir uma vez que as unidades encontram-se isoladas.
3. Ao testar cada unidade primeiro e só depois a sua integração, permite que durante os testes de integração das unidades ou módulos (conjunto de várias unidades) seja mais fácil identificar e corrigir os erros encontrados, tornando a fase de *Integration Testing* mais eficaz.

7.5.3.2 Integration Testing

O nível de teste *Integration Testing* [ver anexo – C5] tem como propósito validar se duas ou mais unidades funcionam correctamente em módulos maiores (conjunto de várias unidades integradas), focando-se em testar ambas as interfaces entre as unidades e os módulos, até que a integração de todos os módulos esteja completa e seja construído um Sistema completo e estável. A metodologia deste nível de teste consiste assim em utilizar como *input* as unidades testadas anteriormente na fase de *Unit Testing,* para de seguida agrega-las em módulos e executar os testes de integração. É importante referir que apesar das unidades já terem (devem) sido testadas anteriormente, é comum encontrar erros durante a fase de integração devido a manipulação inadequada/inexistente de excepções, interfaces incorrectas de comunicação com a base de dados, etc. O objectivo de *Integration Testing* é validar que o Sistema integrado está estável e pronto para a fase de *System Testing*.

Integration Testing possui várias abordagens e estratégias de execução **[37]**:

- Big Bang
- Incremental
    - Top Down
    - Bottom Up

Big Bang

Abordagem simples onde todos os módulos são integrados de uma só vez, e testados. Poderá ser útil em Sistemas pequenos. No entanto para Sistemas de médios ou grandes torna-se difícil a localização dos erros e a equipa teria de esperar bastante tempo até que todos os módulos estivessem terminados para iniciar a fase de testes.

Incremental

Aqui a abordagem resume-se à integração de forma incremental de 2 ou mais módulos de cada vez, que estejam logicamente relacionados, até que todos os módulos sejam integrados e testados com sucesso. Este tipo de abordagem só é possível quando há uma estrutura hierárquica do software, e é essencial o uso dos documentos de design para perceber o fluxo

da aplicação/sistema. Se durante os testes existir a necessidade de comunicação de dados com módulos que ainda não foram integrados, são usados *Stubs (*é chamado pelo módulo que está a ser testado) e *Drivers (*chama o módulo para ser testado) que simplesmente simulam a comunicação de dados com os módulos já integrados. Quando os testes terminam os *Drivers* e *Stubs* são substituídos por módulos reais, até que todos os módulos sejam integrados e testados.

Bottom up

Nesta abordagem cada módulo dos níveis inferiores é testado com os módulos dos níveis imediatamente acima, até que todos os módulos sejam testados. Para ser possível executar os testes, são utilizados *Drivers* para simular os módulos superiores. As vantagens desta abordagem resumem-se ao facto de ser fácil encontrar os erros, e de não ser necessário esperar que todos os módulos estejam concluídos. No entanto é necessário ter em conta que os módulos críticos, usualmente posicionados no topo da arquitectura do software, que controlam o fluxo da aplicação/sistema são testados em último, o que poderá trazer problemas (erros no níveis superiores tendem a propagar-se pelos níveis inferiores). Existe ainda a desvantagem de não ser possível obter um protótipo da aplicação nas fases iniciais, uma vez que os módulos críticos ainda não foram integrados.

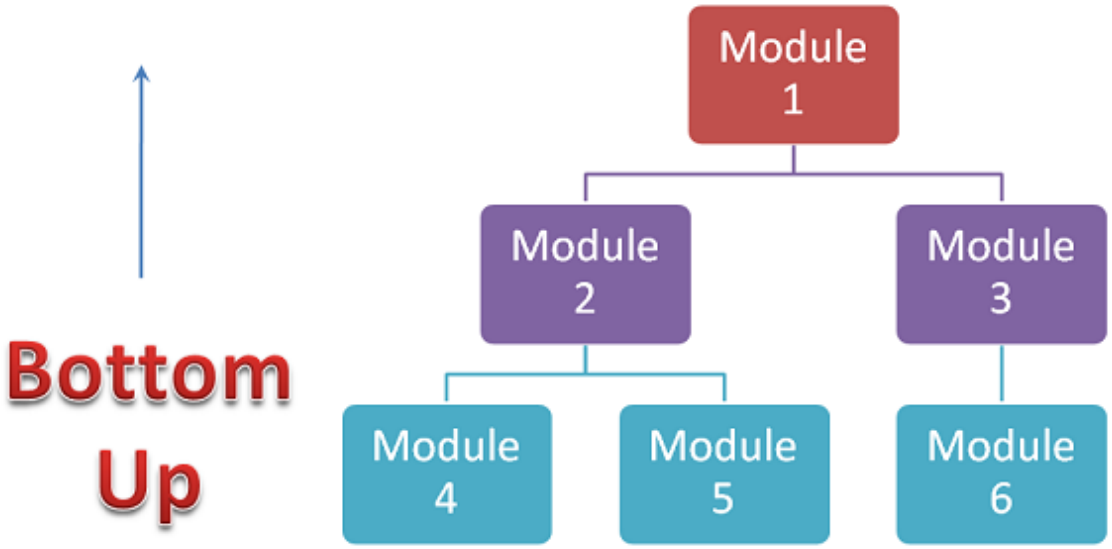


Figura 4 - Integration Bottom up, [guru99]

Top Down

Na abordagem *top down* o *Modus operandi* é o mesmo, com a diferença os testes iniciam-se nos módulos superiores até aos módulos inferiores. As vantagens desta abordagem resumem-se a facilidade de encontrar os erros identificados, os módulos críticos são integrados e testados em primeiro lugar e ainda a possibilidade de obter um protótipo da aplicação nas fases iniciais. A grande desvantagem desta abordagem é a necessidade de serem criados vários *Stubs*, que ao mesmo tempo por serem limitados nas suas capacidades limitam também os testes.

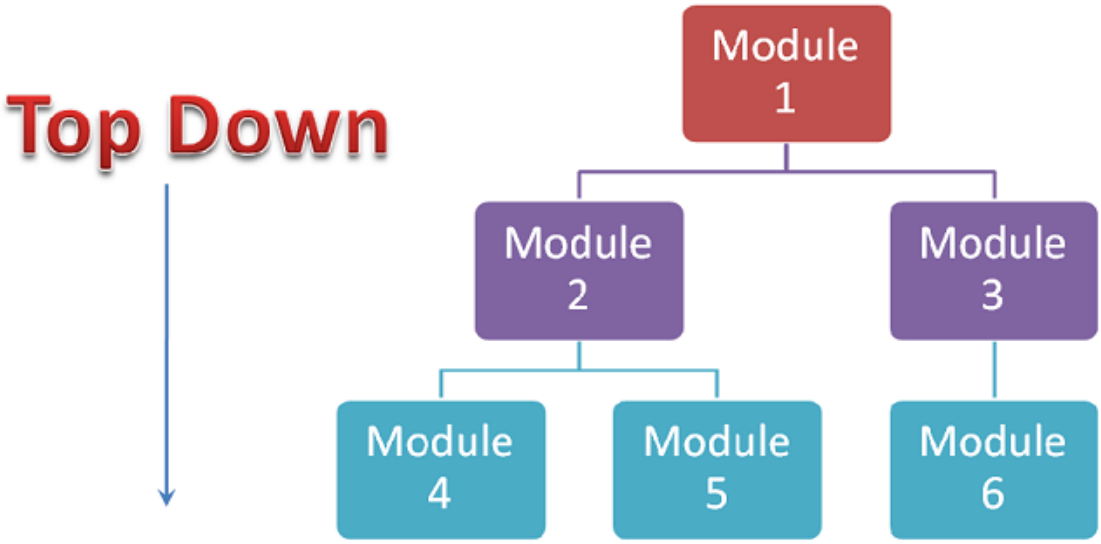


Figura 5 - Integration Top down, [guru99]

7.5.3.3 System Testing

O nível de *System Testing* [ver anexo – C6] pertence ao tipo de teste *Black Box*. O objectivo é validar a qualidade do Sistema como um todo, num ambiente idêntico ao de produção. Esta fase pode ser interpretada como um *test-drive* da aplicação/sistema. Os testes de sistema são baseados nas especificações de alto nível do software, onde o foco é testar a capacidade do Sistema em cumprir as especificações e requisitos (funcionais), medindo e avaliando a qualidade, risco e confiança no software desenvolvido. Contudo os atributos de qualidade não-funcionais (ex.: *reliability, security, etc)* também podem ser validados. Aliás, se o projecto em desenvolvimento pertencer à área de saúde, militar, etc. é provável que os atributos de qualidade não-funcionais sejam testados exaustivamente.

7.5.3.4 Acceptance testing

Acceptance testing [ver anexo – C7] é o último nível de teste. Nesta fase o estado da aplicação/sistema já foi definido como completo por parte da equipa de desenvolvimento, estando assim pronta para ser testada pelos clientes/utilizadores. O objectivo de *Acceptance Testing* não é encontrar erros, mas sim avaliar se o sistema cumpre os requisitos de negócio, junto dos *stakeholders* (clientes/utilizadores), e se está pronto para entrega final. Embora possa parecer redundante testar novamente o software a um nível idêntico ao de *System Testing* (*Acceptance Testing* partilha a mesma abordagem e metodologia, podendo ser considerado como um *subset* do *System Testing*), não é por várias razões:

- Os requisitos podem ter sido alterados durante o desenvolvimento do software, e as alterações não foram comunicadas correctamente para a equipa de desenvolvimento.
- Os requisitos do cliente podem ter sido mal interpretados, e portanto o software não satisfaz por completo as suas necessidades.
- Por vezes, a aceitação dos testes, é usado como forma de fechar e dar por concluído o negócio.

A abordagem para *Acceptance Testing* pode ser divida em 2 tipos **[38]**:

- *Internal Acceptance Testing* – Os testes são executados por colaboradores da empresa que estiveram envolvidos no projecto mas não directamente no desenvolvimento e teste do software, ex.: gestor de vendas, gestor do produto, etc. Uma vez concluída a fase de testes, o relatório de *acceptance testing* poderá então ser enviado e assinado pelo cliente.
- *External Acceptance Testing* –
    - *Customer Acceptance Testing* – Os testes são executados pelo cliente.
    - *User Acceptance Testing* – Os testes são executados pelo utilizador final do Sistema (geralmente conhecido pelos utilizadores como *beta testing).*

A metodologia de *Acceptance Testing* consiste basicamente na análise dos documentos de requisitos, identificação dos cenários de teste e casos de teste necessários, preparar os dados que serão utilizados nos casos de teste e finalmente executar os testes e registar os resultados.

7.6 Monitoring

Na secção 6 de *monitoring* foram definidas uma vez mais as métricas e KPI's necessárias para medir, controlar e avaliar o processo. As métricas incidem sobre os casos de teste, técnica de regressão, esforço (medido em *man-hours*, h:m) das actividades de *software testing* e sobre o tamanho e complexidade do código. As métricas foram definidas de acordo com a maturidade actual da RLS, e tendo em conta um balanceamento saudável entre o esforço para as recolher *vs* benefícios oferecidos. Das métricas especificadas, a sua maioria poderá ser recolhida de forma automatizada através do uso da plataforma JIRA ou das ferramentas de teste [ver anexo – C1] especificadas no processo, assim o *overhead* associado à recolha de métricas poderá ser reduzido de forma significativa. Recolhidas as métricas, as mesmas serão usadas nos KPIs. Os 5 KPIs definidos permitiram medir a performance das actividades de *software testing* nos seguintes aspectos:

1. Cobertura dos casos de teste no software. Os tipos de cobertura escolhidos foram *Statement Coverage* e *Branch Coverage*. Índices altos de cobertura estarão relacionados com melhor qualidade e maior confiança no software em desenvolvimento.
2. Eficácia de execução dos casos de teste. Especialmente em casos onde a metodologia escolhida for do tipo ágil, será importante saber quantos casos de teste a equipa (de forma manual ou automática) consegue executar por *man-hour*. Assim nas iterações seguintes poderá ser estimado com maior precisão o tempo necessário para testes.
3. Erros encontrados durante a técnica de *Regression*. Já mencionado anteriormente a importância desta técnica, é medido o número de casos executados e os respectivos erros encontrados.
4. Erros encontrados por tamanho do código. É um KPI especialmente útil uma vez conjugado com a complexidade do código, que permitirá a RLS estabelecer limites para as unidades de código com o objectivo de reduzir o risco de erros por unidade. O tamanho do código é medido em *KLOC* (1 *KLOC* = 1000 linhas de código; são excluídas linhas com comentários, em branco, etc.). O tipo de erro é utilizado para ajudar a quantificar o risco associado, não tendo sido definidos pesos associados ao tipo de erro (como foi feito no processo *Verification*).
5. Distribuição dos erros. Este KPI poderá ser utilizado para identificar o nível de teste mais eficaz na detecção de erros (especialmente útil se existir pouco tempo disponível para testes), e para identificar possíveis erros nas fases de desenvolvimento do software (*Requirements*, *Design*, *Implementation*, etc.).

7.7 Support Tools

A secção 7 - *Support Tools* diz respeito às ferramentas recomendadas para suportar o processo. A escolha das respectivas ferramentas foi feita após várias pesquisas. A metodologia de pesquisa passou por uma vasta pesquisa com base no Google **[39, 40, 41, 42, 43, 44, 45]** e na interacção com alguns *developers* através de um fórum tecnológico **[46]**. Esta pesquisa inicial deu a capacidade de ficar "por dentro" das ferramentas mais utilizadas na indústria, e ficar a conhecer também os objectivo e funcionalidades de cada uma. Com os conhecimentos necessários, o processo de especificação das ferramentas resumiu-se ao número de funcionalidades, preço e linguagem suportada de cada ferramenta. Sendo que de momento a RLS foca-se maioritariamente em desenvolvimento de software na linguagem JAVA, será possível através das ferramentas especificadas suportar em 100% o processo *Validation*, com a agradável surpresa de não ser necessário adquirir nenhuma ferramenta, uma vez que são todas grátis (com a excepção do *Confluence* e *JIRA* que já foram compradas pela RLS). É importante referir que o uso destas ferramentas terá um papel fundamental na capacidade da RLS em reduzir o o*verhead* associado às actividades de recolha de métricas durante o processo de desenvolvimento de software. Mais ainda, será possível tornar a informação transparente e acessível (ex.: *dashboards*) a todos os colaboradores da RLS para que seja possível avaliar e controlar continuamente a performance e a qualidade nas actividades de *software testing*.

7.8 Tailoring

Na secção 8 de *tailoring* do processo foi definido que as actividades deverão ser alteradas/adaptadas conforme o tipo e necessidades do projecto, ex.: orçamento, linguagem de programação, requisitos críticos (área da saúde, militar, governo, etc.), etc. Assim todas as actividades a desempenhar deverão ser especificadas no *Test Plan*, assim como as actividades que não serão desempenhadas, deverão ser especificadas e justificadas.

Capítulo 8
8.1 Resultados

Como forma de mitigar o risco da RLS em não possuir até ao momento quaisquer métricas de projectos anteriores ou de actividades operacionais (ex.: desenvolvimento de software), para que fosse possível uma análise dos resultados do trabalho desenvolvido, foi tomada a decisão de antecipar a análise de possíveis métricas relevantes [artigo - SEI *Metrics*] para a RLS e defini-las de imediato para o projecto USF que no momento se encontrava na fase inicial de análise de requisitos. O projecto USF é um projecto com orçamento bastante limitado, que tem como objectivo consumir o mínimo de recursos (horas) possíveis. O objectivo durante a análise de possíveis métricas foi identificar um conjunto de métricas que oferecessem a capacidade de controlar os custos do projecto (neste caso medidos em horas) através de:

- Correcto planeamento e controlo do progresso (milestones) do projecto.
- Análise de time tracking das diversas tarefas (incluindo planeamento, reuniões, tarefas de QA, etc.)
- Análise do esforço gasto em actividades de rework.
- Análise da gestão de riscos. Foi identificado pelo Project Leader que o cliente estava constantemente a alterar os requisitos/funcionalidades do projecto. Houve assim a necessidade de definir um conjunto de métricas que medissem o impacto das alterações.
- Análise dos casos de teste. Foram definidas métricas com o objectivo de medir o esforço gasto no planeamento e especificação dos casos de teste, medir o sucesso dos casos de teste e ainda testes de usabilidade onde se pretendia medir o sucesso em desempenhar determinadas tarefas, o tempo demorado e o nível de satisfação (por dificuldade) atribuído pelo utilizador.
- Controlo de manutenção. Como existia a possibilidade de no futuro a RLS ser contratada para fazer a manutenção do Sistema desenvolvido, foram definidas métricas que permitissem medir o número de pedidos de suporte, e o custo (tempo medido em man hours) desses pedidos.
- Esforço de treino. O projecto exigia um período de treino dos utilizadores do Sistema desenvolvido, como tal foram definidas métricas para medir o tempo gasto no treino de cada utilizador.

O projecto foi planeado no JIRA e tinha como objectivo registar a maior parte das métricas directamente no JIRA. Para além das métricas definidas e com visibilidade sobre os processos (V&V) a implementar, foram definidos alguns métodos de verificação e validação para o projecto:

- Verificação dos requisitos –Walkthrough, com a possibilidade do cliente estar presente.
- Validação dos requisitos – Acceptance testing através de mockups.
- Verificação da arquitectura – Walkthrough.

No entanto até o momento não foi possível recolher qualquer tipo de métricas do projecto uma vez que o Project Leader, inesperadamente para a RLS, teve de ser atribuído a outro projecto. Apesar de não ser possível demonstrar os resultados das métricas e métodos V&V definidos, voltando a analisar as métricas especificadas, já com conhecimentos significativamente mais profundos, para o projecto USF é possível encontrar um erro que muito provavelmente iria ser detectado logo nos primeiros registos das métricas. O erro está

no número de métricas definidas. Foram definidas demasiadas métricas que muito provavelmente iriam causar overhead nas actividades do projecto, entrando assim em conflito com um dos principais objectivos do projecto de manter os custos (tempo) o mais baixos possível para a RLS.

8.2 Caso Prático Verfication

No processo *Verification* houve a oportunidade de recolher resultados de um caso prático. Nomeadamente depois de definido o procedimento *Documents Review*, o mesmo foi utilizado para verificar o próprio procedimento. A primeira tentativa de revisão do documento consistia numa *Inspection*, que no entanto foi cancelada pelo moderador por não cumprir as condições de entrada. Como tal a primeira revisão consistiu em 3 revisões individuais (ad-hoc), que na verdade resultaram da fase de preparação de cada revisor devido à *Inspection* cancelada (a *Inspection* foi cancelada durante a fase de preparação apenas).

Após o *rework* das primeiras 3 revisões, o procedimento *Documents Review* foi de novo submetido a uma Inspection. Desta vez com sucesso [ver anexo – F1], a equipa da *Inspection* foi formada pelo autor, que desempenhou o papel de *recorder*, por 3 revisores e por 1 moderador. A fase de preparação da *Inspection* registou 5h45m (soma do tempo de cada revisor + moderador), tendo a reunião demorado 2h10m. Tendo em conta o tamanho do artefacto, 20 páginas, as *Best practices* estiveram perto da realidade. A reunião na verdade demorou mais de 90minutos, mas foi divida em duas, tenho sido feito um intervalo após os primeiros 90minutos, e os valores da velocidade de revisão estiveram dentro dos limites estabelecidos pelas *Best practices*.

Quanto aos erros foram encontrados 29 no total, sendo que 27 foram definidos como tipo médio e 2 como críticos. Após a fase de *rework* do autor ter sido verificada pelo moderador, o procedimento foi aprovado pela RLS e publicado no seu QM. Apesar do custo da reunião (5h45m + 2h10m*'4 colaboradores') a RLS teve a capacidade de publicar no seu QM um procedimento importante, sabendo que não contém nenhum erro por identificar, mas apenas possíveis melhorias contínuas.

Até a data ainda não foram realizadas mais *Inspections* nem *Walkthroughs* pelo motivo que neste momento não é possível ter disponível o *staff* necessário para as desempenhar. No entanto o procedimento continuou a ser aplicado, tendo sido feitas várias inspecções do tipo *Ad-Hoc*, sempre respeitando os critérios definidos pelo procedimento.

Com o processo de *Verification* surgiu também a primeira oportunidade de eliminar *overhead* com origem no simples facto dos documentos serem controlados não só quando criados mas também quando actualizados. Em empresas tradicionais a necessidade de se manter manualmente as versões dos documentos actualizadas, de controlar constantemente as alterações dos documentos feitas pelos colaboradores, e notificar os responsáveis para a necessidade de uma nova revisão a esses documentos actualizados, são factores que podem determinar à partida o insucesso da implementação do processo *Verification* pelo respectivo acréscimo de *overhead*. Na RLS foi possível eliminar o *overhead* do processo *Verification* através da implementação de um sistema de gestão documental, que possibilita o controlo automático das versões dos documentos, que permita desenhar *workflows* com diferentes estados para documentos (*Draft, Ready, Reviewed e Published),* que ofereça a capacidade de comparar as alterações feitas entre versões e sobretudo que permita notificar de forma automática e em tempo real os responsáveis, sempre que exista uma condição que inicie o processo *Verification*, ou seja, sempre que um documento é criado ou alterado para uma nova versão. Este sistema foi implementado recorrendo ao uso do *Confluence* e do plugin *Ad-Hoc* Workflows para o

Confluence. O sistema implementado satisfaz ainda os requisitos e necessidades da empresa em corresponder aos requisitos ISO 9001:2008, nomeadamente do controlo documental.

8.3 Conclusões

Em relação à primeira fase do estágio, esta revelou-se um grande desafio e superou bastante as minhas expectativas pela importância, rigor e nível de detalhe que a área de Engenharia de Software exige quando correctamente aplicada. Uma das dificuldades durante este primeiro semestre residiu na actividade de análise de requisitos, onde foram necessárias várias horas de trabalho. No entanto tive oportunidade de perceber como fazer uma correcta análise através do uso de diversos métodos aplicados num contexto real. Foi-me dada ainda a oportunidade de participar em actividades de especificação do QMS da RLS. Estas actividades foram bastante enriquecedoras a nível pessoa, pois permitiram participar na análise e resolução de problemas reais.

O estágio sofreu ainda várias oscilações no foco do plano de estágio, uma vez que apenas lentamente se foi percebendo a verdadeira dimensão e complexidade de todo o projecto Athena, até que foi necessário definir novamente um plano de estágio. Razão pela qual foi necessário terminar o estágio apenas na fase especial. O novo plano de estágio partiu da minha iniciativa pessoal, mesmo estando consciente dos riscos e dificuldades associadas. O gosto por Engenharia de Software e a vontade de adquirir um conjunto de conhecimentos complementares ao percurso até então percorrida na Universidade foram determinantes. No entanto o estágio foi de alta dificuldade, mais do que esperava para ser sincero, o que me fez crescer também a nível pessoal para conseguir ultrapassar todos os obstáculos e sobretudo a falta de conhecimento na área de Engenharia de Software e Qualidade.

Sendo o desenvolvimento de software um dos maiores mercados da actualidade mas possuindo ao mesmo tempo vários problemas de gestão e planeamento, este estágio deu-me a oportunidade única de adquirir conhecimentos, que não são comuns para um recém-licenciado da área, que pretendo utilizar para procurar oportunidades de grandes desafios e responsabilidades. Vejo-me no futuro próximo a procurar obter um conjunto de certificados (ex.: SCRUM Master, Project Management Professional) de gestão e planeamento, que me possibilitem continuar a desenvolver os conhecimentos já inicializados durante o estágio na RLS, para que possa perseguir uma carreira que não seja exclusiva para o desenvolvimento de software, mas também de Engenharia de Software.

Referências

[1] Available from: http://www.iso.org/iso/home.html; accessed 4 Dezembro 2012.

[2] Available from: http://en.wikipedia.org/wiki/Software_quality; accessed 10 Dezembro 2012.

[3] Available from: http://en.wikipedia.org/wiki/Software_quality; accessed 10 Dezembro 2012.

[4] Available from: http://www.mindtools.com/pages/article/newSTR_75.htm; accessed 8 Dezembro 2012.

[5]Available from: http://en.wikipedia.org/wiki/History_of_software_engineering; accessed 10 Dezembro 2012.

[6]Available from: http://www.heinz.cmu.edu/project/india/pubs/qualifier1.pdf; accessed 10 Dezembro 2012.

[7] Hoyle, D. *ISO 9000 Quality System Handbook.* Elsevier Ltd, Oxford, 2009.

[8] Available from: http://www.iso-9001-checklist.co.uk/iso-9001-statistics.htm; accessed 14 Dezembro 2012.

[9] Maria, C. *Mapeando ISO 9001 para CMMI,* Fortaleza 2007.

[10] Available from: http://www.emeraldinsight.com/content_images/fig/0510180405003.png; accessed 14 Dezembro 2012.

[11] Available from: http://bls.gov/fls/images/chartbook3.2.png; accessed 14 Dezembro 2012.

[12] Available from: http://www.sei.cmu.edu; accessed 19 Dezembro 2012.

[13] Available from: http://www.tutorialspoint.com/cmmi/cmmi-models.htm; accessed 19 Dezembro 2012.

[14] Available from: http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/82.78.pdf; accessed 19 Dezembro 2012.

[15]Available from: http://www.sw-engineering-candies.com/blog-1/rules-of-thumb-in-software-engineering; accessed 8 Janeiro 2012.

[16] Available from: http://www.blogcmmi.com.br/avaliacao/como-anda-o-cmmi-no-mundo-2011; accessed 9 Janeiro 2012.

[17] Available from: http://www.umsl.edu/~sauterv/analysis/F08papers/Jakrapan_CMMI_and_ISO_Success.html#ISO; accessed 15 Janeiro 2013.

[18] Available from: http://www.srtsolutions.com/software-the-most-important-growth-industry; accessed 15 Janeiro 2013.

[19] Available from: http://online.wsj.com/article/SB10001424053111903480904576512250915629460.html; accessed 15 Janeiro 2013.

[20] RedLight Software

[22] Fagan, M.E., Advances in Software Inspections, July 1986, IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, Page 744-751

[23] Available from: http://www3.dsi.uminho.pt/rmac/privatefiles/papers/2009_SEDES_MonteiroMachadoKazman-ieee.pdf; accessed 15 Junho 2013.

[24] Available from: http://www.bioinfo.in/uploadfiles/13111284371_1_1_BIOINFO_SC.pdf; accessed 15 Junho 2013.

[25] Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.2351&rep=rep1&type=pdf; accessed 15 Junho 2013.

[26] Available from: https://cs.uwaterloo.ca/~dberry/COURSES/software.engr/lectures.pdf/inspect.pdf; accessed 15 Junho 2013.

[27] Available from: http://www.osel.co.uk/fitsbnwtf.pdf; accessed 15 Junho 2013.

[28] Available from: Software Testing guide book.pdf; accessed 15 Junho 2013.

[29] Available from: http://searchsoftwarequality.techtarget.com/definition/gray-box; accessed 25 Julho 2013.

[30] Available from: http://softwaretestingfundamentals.com/gray-box-testing/; accessed 25 Julho 2013.

[31] Available from: Different Approaches to White Box Testing Technique for Finding Errors.pdf; accessed 25 Julho 2013.

[32] Available from: http://homepage.hispeed.ch/pjcj/testing_and_code_coverage/paper.html; accessed 25 Julho 2013.

[33] Available from: http://testingwarrior.blogspot.pt/2011/12/cyclomatic-complexity-with-example.html; accessed 25 Julho 2013.

[34] Available from: http://www.guru99.com/cyclomatic-complexity.html; accessed 25 Julho 2013.

[35] Available from:http://www.klocwork.com/products/documentation/current/McCabe_Cyclomatic_Complexity; accessed 25 Julho 2013.

[36] Available from: http://blogs.msdn.com/b/zainnab/archive/2011/05/17/code-metrics-cyclomatic-complexity.aspx; accessed 25 Julho 2013.

[37] Available from: http://www.guru99.com/integration-testing.html; accessed 25 Julho 2013.

[38] Available from: http://softwaretestingfundamentals.com/acceptance-testing/; accessed 25 Julho 2013.

[39] Available from:http://onlysoftware.wordpress.com/2012/12/19/code-coverage-tools-jacoco-cobertura-emma-comparison-in-sonar/; accessed 25 Julho 2013.

[40] Available from: http://blogs.sourceallies.com/2012/03/code-quality-metrics-with-sonar-part-i/; accessed 25 Julho 2013.

[41] Available from: http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis; accessed 25 Julho 2013.

[42] Available from: http://www.javacodegeeks.com/2012/10/java-code-quality-tools-overview.html; accessed 29 Julho 2013.

[43] Available from: http://java.dzone.com/articles/code-quality-tools-java; accessed 29 Julho 2013.

[44] Available from: http://zeroturnaround.com/rebellabs/developer-productivity-report-2012-java-tools-tech-devs-and-data/; accessed 29 Julho 2013.

[45] Available from: http://zeroturnaround.com/rebellabs/code-quality-tools-review-for-2013-sonar-findbugs-pmd-and-checkstyle/; accessed 29 Julho 2013.

[46] Available from: http://forum.zwame.pt/; accessed 1 Junho 2013.

[47] Available from: http://www.information-management.com/; accessed 29 Julho 2013.

[48] Available from: http://www.dtic.mil/ndia/2005cmmi/monday/olson.pdf; accessed 29 Julho 2013.