Faculdade de Engenharia da Universidade do Porto

Desenvolvimento de *Software* para Dispositivos Médicos: Uma Abordagem Ágil

Manuel Zamith

Mestrado em Engenharia de Software
Porto, Portugal
Junho 2018

Supervisor: Professor Doutor Gil Gonçalves

Agradecimentos

Esta dissertação representa o culminar de uma segunda aventura pelo mundo académico. Desta vez completamente direcionada para os meus objetivos profissionais e para uma procura por competências que, outrora, se encontravam completamente fora da minha zona de conforto. É por isso que gostaria de agradecer em primeiro lugar à Critical Software e à ITGrow Software e Sistemas por me terem proporcionado a oportunidade de frequentar o Mestrado em Engenharia de Software numas condições privilegiadas. Não só por permitirem que frequentasse as aulas do mestrado sem qualquer restrição, mas também pelo apoio incondicional e interesse que demonstraram no decorrer de todo este percurso.

Um grande agradecimento ao Professor Gil Gonçalves, orientador desta dissertação, por me ter apresentado este desafio, à medida daquilo que eram os meus interesses pessoais e a minha situação profissional. Obrigado pelas palavras de sabedoria e por não ter colocado qualquer barreira para que pudesse aprofundar os temas que mais me interessavam.

Agradeço também ao restante corpo docente da FEUP que teve a coragem de iniciar este curso, apostando num conjunto de alunos com um leque de competências tão diversificadas. Alunos estes, meus colegas de turma, aos quais também dirijo uma palavra de gratidão. Somos os pioneiros, nunca se esqueçam.

À Mónica Sobreira, ao Patrick Machado e ao Patrício Correia, que ocupam posições de liderança no projeto onde me insiro há já três anos, obrigado não só por terem contribuído ativamente no trabalho desenvolvido nesta dissertação, mas também por todos os ensinamentos aos quais já me habituaram diariamente.

Um agradecimento também ao Vítor Vieira, à Mariana Madureira e aos elementos da equipa *Prehab* da edição de 2018 de LGP pelos seus contributos para este trabalho.

Aos meus pais e irmão, qualquer agradecimento é pouco. Qualquer que seja a aventura, vocês embarcam na viagem comigo. Obrigado por me protegerem e por compreenderem todas as minhas falhas.

Por último, mas nunca menos importante (mais uma vez), a ti Rita, que nunca deixas de me surpreender pela tua resiliência. Já te obriguei a sobreviver a duas teses, espero que seja a última. «Obrigado por me salvares e por não desistires de mim. Se concluí esta dissertação, a ti o devo.».

Thesis advisor: Professor Doutor Gil Gonçalves Manuel Zamith

Medical Device Software Development: An Agile Approach

Abstract

Regulation of medical devices has been one of the most prominent initiatives of the European Union in the health domain. The recent Medical Device Regulation 2017/745/EEC extended the definition of medical devices to standalone software systems with prognostic and prevision intended purposes. This paradigm shift stimulates the development of more lightweight software systems such as mobile applications, that can be classified as legitimate medical devices and can be prescribed to patients. This new context creates the motivation for an urgent adjustment of the currently used software development life cycle models and processes.

Aditionally, agile development offers a lightweight framework for helping teams maintain a focus on the rapid delivery of business value. As a result of this focus, the benefits of agile software development are that organizations are capable of significantly reducing the overall risk associated with software development, given a constantly evolving functional and technical landscape.

This dissertation discusses a tailored agile approach based on the SCRUM model, designed to be compliant with the international standards for medical device software development and benefit the creation of software solutions according to the current Medical Device Framework. The discussion in this thesis demonstrates there is no reason to believe that agile methodologies should not benefit the process of creating software solutions in the medical device domain.

These conclusions were supported by professionals in the software development industry with experience working in critical systems, as well as qualified individuals working in the medical device field.

Orientador: Professor Doutor Gil Gonçalves Manuel Zamith

Desenvolvimento de *Software* para Dispositivos Médicos: Uma Abordagem Ágil

Resumo

A regulação dos dispositivos médicos tem sido uma das iniciativas mais proeminentes da União Europeia relativamente à indústria médica. O recente Regulamento para os Dispositivos Médicos 2017/745/EEC veio estender a definição de dispositivo médico para sistemas de *software* isolados com finalidades previstas de previsão e diagnóstico. Esta mudança de paradigma promove o desenvolvimento de sistemas de *software* mais dinâmicos e de menos dimensão e complexidade como aplicações móveis, que podem ser classificadas legitimamente como dispositivos médicos e, ultimamente, prescritas a pacientes. Este novo contexto cria uma necessidade urgente de ajustar os modelos de ciclo de vida de desenvolvimento de *software* utilizados presentemente na indústria.

Adicionalmente, o desenvolvimento *agile* oferece ferramentas básicas para ajudar equipas a manter o foco na entrega rápida de valor de negócio. Como resultado, os benefícios das metodologias ágeis para desenvolvimento de *software* resultam na redução do risco associado ao processo em ecossistemas de rápida evolução tecnológica e funcional.

Esta dissertação discute uma abordagem *agile* modificada, baseada no modelo *SCRUM*, desenhada para cumprir com as recomendações internacionais para o desenvolvimento de dispositivos médicos e estimular um desenvolvimento mais eficiente de *software*, tendo presente as recentes mudanças no *Medical Device Framework*. A discussão apresentada demonstra que não existem motivos para fundamentar a ideia de que as metodologias ágeis não beneficiam o processo de criar soluções de *software* no domínio dos dispositivos médicos.

Estas conclusões foram suportadas por profissionais da indústria do desenvolvimento de soluções críticas, assim como indivíduos credenciados na indústria dos dispositivos médicos, quer nacional, quer internacionalmente.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos

CE	Comissão Europeia
DDM	Diretiva para Dispositivos Médicos
DAS	Documento de Arquitetura de *Software*
DER	Documento de Especificação de Requisitos
DoD	*Definition of Done*
DPS	Documento de Planificação de *Software*
EC	*European Commission*
GC	Gestão de Configurações
IDE	*Integrated Development Environment*
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electrical and Electronics Engineers*
INFARMED	Instituto Nacional da Farmácia e do Medicamento
ISO	*International Organization for Standardization*
LGP	Laboratório de Gestão de Projetos
LIC	Lista de Itens de Configuração
MDF	*Medical Device Framework*
ON	Organismos Notificados
PIT	Plataforma de *Issue Tracking*
QA	*Quality Assurance*
RDM	Regulamento para Dispositivos Médicos
SCV	Sistema de Controlo de Versões
SGQ	Sistemas de Gestão da Qualidade
TDD	*Test-Driven Development*
TI	Tecnologias da Informação
UE	União Europeia
UML	*Unified Modeling Language*
XP	*Extreme Programming*

1 Introdução

Em Engenharia de Software, podemos descrever sistemas críticos como:

> *«Sistemas cuja falha pode resultar em perda de vida, dano significativo da propriedade ou dano ao ambiente.»* [1]

Trata-se uma definição bastante intuitiva, que classifica formalmente um sistema de acordo com as consequências da sua falha. Tradicionalmente, esta definição engloba indústrias como a militar, a da aviação, a dos transportes, a nuclear e também a indústria médica [1, 2].

Existem vários casos massivamente documentados de incidentes relacionados com falhas de sistemas críticos. Destaco, a título de exemplo, o *Mars Climate Orbiter* que, em 1998, falhou a missão de entrar na órbita do planeta Marte devido a um erro de cálculo relacionado com unidades de medida [3]. Estes incidentes podem ser especialmente perigosos quando ligados à indústria médica [4].

Entre 1985 e 1987, uma máquina de radioterapia computorizada deu origem a um dos acidentes mais citados, relacionados com *software* e sistemas críticos. O *Therac-25*, considerado moderno à luz da

época, inovava na medida em que permitia a administração de doses reguláveis de radiação através do mesmo equipamento. Contudo, erros no funcionamento da máquina levaram a 6 acidentes nos quais os pacientes receberem uma overdose de radiação, originando um total de 5 mortes [5]. Os incidentes do *Therac-25* são atribuídos pela bibliografia, em grande parte, a falhas no processo de desenvolvimento de *software*, nomeadamente no que diz respeito à construção de documentação, ao processo de testes e validação e à fase de desenho e arquitetura [5].

A engenharia de sistemas críticos é uma tarefa complexa e que envolve aspetos técnicos das áreas mais diversas. Por este motivo, e tendo em mente os casos mencionados anteriormente, é fácil perceber a importância do processo de engenharia de *software* no desenvolvimento de sistemas *safety-critical*. Em particular, a indústria médica apresenta um conjunto de desafios únicos caracterizados pela necessidade de garantir sistemas seguros e de confiança. A próxima secção tem como objetivo abordar este tema.

1.1 Desenvolvimento de Software na Indústria Médica

A presença de sistemas de *software* no funcionamento da indústria médica, assim como a sua dependência em relação aos mesmos, é inegável. Mais de 50% dos dispositivos médicos existentes dependem do funcionamento de sistemas de *software* de alguma forma [6]. A indústria das tecnologias da informação (TI) no setor médico tem-se revelado, também, um negócio lucrativo. Já em 2005, o negócio gerado pela mesma rondou os 100 biliões de dólares, valor comparável em escala à economia da América Latina no mesmo ano [7].

Todas as tecnologias que rodeiam o setor médico, desde os dispositivos médicos aos sistemas de TI, mostram um enorme potencial para proporcionar uma mudança positiva para os pacientes, para as suas famílias e para os profissionais de saúde, levando a melhorias como diagnósticos mais rápidos e menos erros clínicos. Como demonstração deste potencial e do espírito inovador existente, podemos constatar que o ramo da tecnologia médica registou mais de 10.000 patentes (de um total de menos de 100.000), no ano de 2012, na Europa, de longe a indústria mais inovadora no continente [8]. Contudo, é um setor que se caracteriza pelos desafios únicos que apresenta. O risco associado à sua falha, bem como a sua crescente complexidade, são alguns dos fatores que a indústria do desenvolvimento tem de estar preparada para enfrentar [6].

Globalmente, o estado atual dos sistemas de informação nos serviços de saúde e hospitais pelo mundo fora é primitivo, e considerado por muitos um excelente exemplo de como as características da área médica podem apresentar desafios difíceis de ultrapassar no que diz respeito ao desenvolvimento tecnológico [7, 9]. Especificamente, comparando com a indústria dos serviços financeiros, do retalho ou da fabricação, é justo constatar um atraso significativo. Este aspeto não é um resultado da incompetência por parte de gestores e distribuidores presentes no setor, mas sim da exigência e complexidade dos serviços e produtos

de saúde.

O grau de variabilidade e incerteza existente ao nível dos serviços de saúde é responsável pela dificuldade enorme de adaptação das tecnologias de informação neste setor. De acordo com Peter Druncker, um dos pais da gestão moderna:

> *"The hospital — altogether the most complex human organization ever devised..."* [10]

Não existe um inventário padronizado para os serviços de saúde; em detrimento, estes têm de ser personalizados no ponto de atendimento, em tempo real e mediante as circunstâncias [7]. A enumeração de um conjunto finito de casos de uso é uma tarefa destinada ao insucesso.

Adicionalmente, existem diferenças significativas na anamnese* e resposta clínica perante diagnósticos incertos em cada médico. Os fatores de influência incluem as características sócio-demográficas, as experiências e as crenças pessoais, o historial clínico do paciente, a especialidade do médico, os recursos do hospital ou unidade de saúde, entre outros [7]. Esta variabilidade cria problemas nos pontos de colisão que existem entre *stakeholders* (médicos, enfermeiros, técnicos de saúde e auxiliares de ação médica). Coloque-se o cenário ilustrado pela figura 1.1: um paciente apresenta-se numa unidade de saúde, onde discute uma história de intolerância alimentar com o seu médico de medicina geral e familiar. Este prescreve uma endoscopia digestiva alta, que o paciente decide realizar numa unidade de exame de diagnóstico privada, com um gastroenterologista. Na consulta seguinte, ao verificar as alterações no relatório do exame, o médico encaminha o paciente para o hospital, onde é observado no serviço de urgência e, posteriormente, internado no serviço de cirurgia geral. Compreende-se que existem, neste cenário, múltiplos pontos de colisão entre os serviços prestadores de cuidados de saúde e, nestas colisões, reside o potencial para a anarquia [7].

Cada profissão e especialidade tem um ponto de vista único da sua centralidade para o processo, a sua linguagem e a sua visão. Ainda mais importante: cada profissão terá, potencialmente, o seu próprio sistema de informação. Em muitos hospitais e clínicas, isto significa que para cada utente poderá haver um número incerto de registos, espalhados por diversos departamentos clínicos e administrativos. Esta fragmentação da informação clínica espelha a fragmentação no processo de prestação de serviços de saúde, independentemente do mecanismo que o financia.

Existem ainda outros desafios que são característicos nesta indústria. Os custos associados aos cuidados de saúde são um fardo para o orçamento de qualquer estado europeu e aquilo que se chama de *high-tech medicine* é encarado como um veículo de mais despesa. Contudo, é possível argumentar que a tecnologia inovadora também será algo necessário para dinamizar a poupança de recursos, potenciando tratamentos mais rápidos e mais eficientes [8].

*Anamnese (do grego *ana*, trazer de novo e *mnesis*, memória) é uma entrevista realizada pelo profissional de saúde ao seu doente, que tem a intenção de ser um ponto inicial no diagnóstico de uma doença ou patologia.

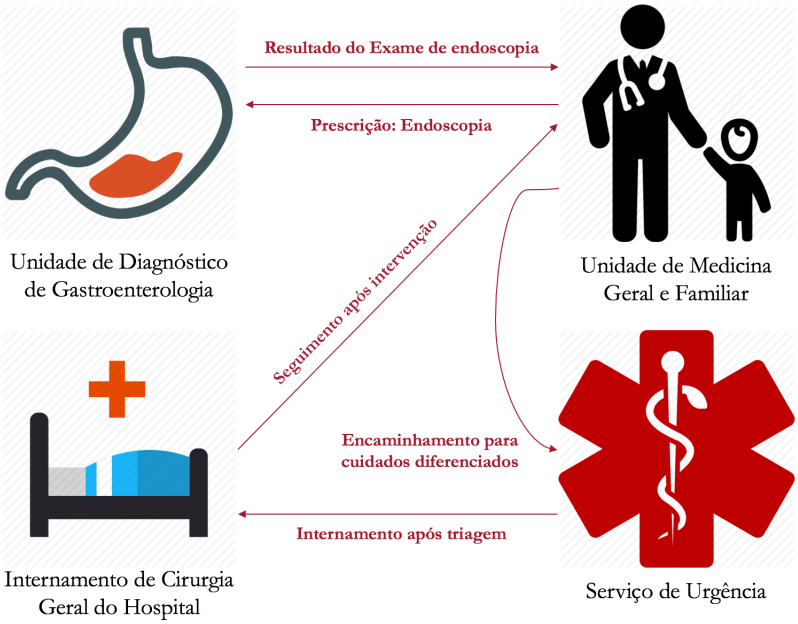


Figura 1.1: Fluxo exemplificativo da troca de informação entre múltiplos sistemas característica do setor da saúde.

Para além disto, a indústria médica caracteriza-se, também, pela forte influência de entidades de regulamentação no que diz respeito ao desenvolvimento de dispositivos médicos e sistemas de informação. O desenvolvimento de sistemas neste setor está sujeito a processos de certificação e testes de conformidade a um conjunto de padrões pré-estabelecidos e fortemente restritivos. Motivados pela necessidade de evitar riscos para os pacientes e para os profissionais de saúde, as autoridades reguladoras exigem especial atenção em todos os passos, tanto de desenvolvimento como da manutenção dos sistemas e equipamentos. De forma a *sobreviver* ao trajeto, desde o desenvolvimento até à aprovação pelas autoridades competentes, um dispositivo médico necessita de ser submetido a uma série de procedimentos que visam garantir a segurança de todos os utilizadores do produto. Para obter aprovação, é necessário implementar sistemas de qualidade, gestão de risco e ter um processo de engenharia que esteja em conformidade com as normas reconhecidas internacionalmente [11].

É importante reforçar que a supervisão regulatória governa não só o desempenho do produto, mas também como este produto é desenvolvido. Existe a forte convicção de que um processo de engenharia sistemático e com uma forte planificação é capaz de produzir dispositivos e sistemas mais confiáveis [12].

A ideia de que os produtos que compramos e usamos no nosso dia-a-dia são regulados e controlados de forma a garantir que são seguros para serem utilizados é algo que a maioria dos indivíduos encara como garantido. Quando compramos roupa, mobília ou brinquedos, existe a expectativa razoável de que cumpram determinados requisitos relacionados com a segurança da sua utilização. Em relação aos

dispositivos e objetos intervenientes nos os nossos cuidados de saúde (incluindo *software*) a necessidade de segurança é particularmente fundamental [13]. Da perspetiva do paciente, há a suposição de que todo o tipo de sistemas utilizados cumprem um conjunto de *standards* minuciosos de qualidade. Por outro lado, da perspetiva dos profissionais de saúde, é imperativo que estes critérios de qualidade e segurança sejam garantidos, de forma a evitar que sejam responsabilizados legalmente no caso de acontecerem incidentes com os cuidados prestados ao paciente, mas também de forma a cumprirem as suas obrigações éticas e profissionais.

A assistência médica e os cuidados de saúde ficarão menos intrusivos e mais eficientes. Novas tecnologias associadas ao desenvolvimento de dispositivos médicos e sistemas de informação irão destruir barreiras ao acesso de conhecimento e vencer a complexidade das organizações de saúde, tanto de uma forma regional, como mundial. Os serviços de saúde tornar-se-ão mais *inteligentes*, adaptáveis e contextualizados dentro do ecossistema da vida humana [7]. Estes progressos serão uma consequência não só da evolução tecnológica computacional, mas também da modernização dos processos de engenharia, nomeadamente da engenharia de *software* [14].

1.2 Motivação

As soluções tecnológicas mostram, inegavelmente, grande potencial para afetar positivamente a experiência na área da saúde, quer para profissionais quer para utentes. Alguns exemplos incluem diagnósticos mais rápidos e precisos, menos erros médicos e menos tempo desperdiçado com burocracias e informação duplicada. Contribuições feitas para a melhoria da qualidade do *software* na indústria médica são contribuições que ultimamente subscrevem a melhoria dos cuidados de saúde.

Em 2007, através da diretiva 2007/47/EC [15] publicada pelo Conselho Europeu, introduziu-se uma mudança radical na definição legal de dispositivo médico na União Europeia, na medida em que o *software* passou a ser possivelmente considerado um dispositivo médico de ser livre direito, sem necessitar de estar embebido num dispositivo eletrónico. Mais recentemente, no ano de 2017, entrou em vigor o Regulamento 2017/45/EEC [16] que estende esta definição para dispositivos de prognóstico e previsão, consolidando o potencial para as soluções de *software* se estabelecerem como dispositivos médicos preponderantes na indústria.

Estas mudanças no panorama legal europeu e nacional serão abordadas com detalhe no capítulo 2 e têm consequências radicais para a comunidade da engenharia de *software* concentrada no desenvolvimento de dispositivos médicos. É possível, agora, desenvolver soluções de *software* de um nível bastante mais elevado de abstração do que é hábito na indústria médica como, por exemplo, aplicações móveis. Estas aplicações poderão ser certificadas e prescritas pelos profissionais de saúde aos seus doentes e, em último caso, comparticipadas pelo estado à luz das contingências legais em vigor.

No que diz respeito ao processo de engenharia, nomeadamente ao ciclo de vida de desenvolvimento, as consequências também são assinaláveis. Os modelos mais apropriados para o desenvolvimento de uma determinada solução são dependentes das características tecnológicas da mesma. Desenvolver software embebido para um dispositivo eletrónico é radicalmente diferente de desenvolver uma aplicação para *smartphone*. Adicionalmente, as normas internacionais não estão atualizadas neste aspeto, na medida em que o *standard* em efeito é a IEC 62304:2006 [17], que parte do pressuposto que o *software* é um subsistema de outro produto. Além disso, esta norma é baseada na ISO/IEC 12207:1995 [18] (norma mais genérica com orientações para os processos de ciclo de vida de desenvolvimento), norma esta que já foi revista duas vezes, em 2008 e 2017.

1.3 Hipótese de Investigação

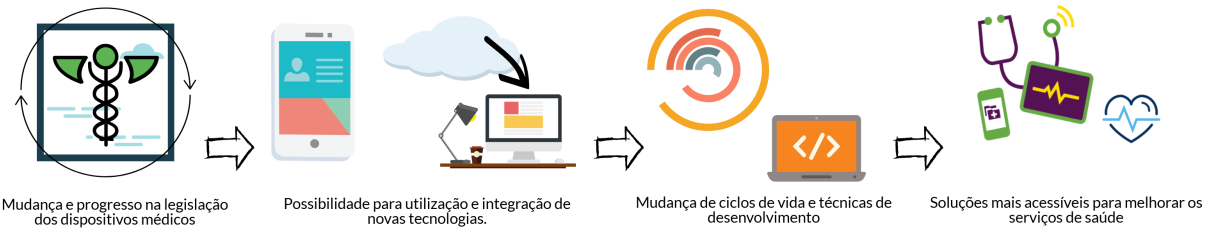


Figura 1.2: Ilustração da hipótese de investigação subjacente a esta dissertação.

Partindo daquilo que foi mencionado na secção anterior, é sensato concluir que se avizinham mudanças no que diz respeito à criação de produtos de *software* aplicado ao desenvolvimento de dispositivos médicos. É urgente que os processos utilizados na indústria se adaptem à nova realidade europeia e endereçem as novas potencialidades emergentes.

A figura 1.2 pretende ilustrar a hipótese de investigação na qual se concentra esta dissertação. As mudanças no panorama legal europeu trazem potencialidades novas para o desenvolvimento de *software* que relevam um problema para a indústria, na medida em que os modelos de desenvolvimento que usam necessitam de se adaptar a esta nova realidade. Pretende-se dar uma resposta a este problema, discutindo um novo ciclo de vida desenhado para potenciar um desenvolvimento mais flexível e produtivo, mas que ainda assim consiga ir de encontro as requisitos impostos pela legislação aplicável aos dispositivos médicos, bem como aos *standards* internacionais existentes. Desta forma, será possível fazer com que estas soluções de *software* cheguem às mãos dos profissionais de saúde e dos seus pacientes para que possam aumentar a sua qualidade de vida.

1.4 Estrutura do Documento

Segue-se uma breve explicação da estruturação deste documento.

No presente capítulo apresenta-se um enquadramento do ecossistema que é o desenvolvimento de *software* na área da saúde e dos sistemas críticos em geral, assim como os seus principais desafios que serviram de motivação para a redação deste documento;

Deste ponto em diante, a dissertação está conceptualmente dividida em três partes distintas. A primeira, do qual constam os quatro primeiros capítulos, pretende aglomerar a o tralho de pesquisa bibliográfica efetuado sobre as principais temáticas relevantes para a investigação realizada.

No capítulo 2 visita-se o enquadramento legal que rege os dispositivos médicos na União Europeia, bem como o conjunto de *standards* e recomendações internacionais para o desenvolvimento de *software* aplicado a dispositivos médicos. Além disso, explicam-se as mais recentes alterações que potenciam o destaque das soluções de *software* neste contexto.

Já no capítulo 3 fala-se no processo de engenharia de *software* e nos principais modelos do ciclo de desenvolvimento. É também feito um comentário sobre as vantagens e desvantagens de cada modelo, sugerindo uma abordagem que tire partido das melhores características de cada um.

No capítulo 4 é discutido o estado da arte relativamente ao processo de desenvolvimento de soluções de *software* aplicado à área da saúde, bem como os respetivos modelos de ciclo de vida, com especial incidência nas metodologias *agile*.

Na segunda parte da dissertação, que engloba os dois capítulos seguintes, discutem-se as soluções propostas para o problema em questão.

No capítulo 5 apresenta-se um novo ciclo de vida de desenvolvimento de *sofware* desenhado para permitir o cumprimento das recomendações apresentadas na norma IEC 62304:2006 e simultaneamente tirar partido das vantagens das metodologias ágeis.

Para complementar esta temática, no capítulo 6 destacam-se as particularidades da norma em questão, assim como estratégias para superar as exigências feitas pela mesma.

A terceira parte deste trabalho concentra-se na validação das propostas apresentadas. No capítulo 7 explica-se de que forma foi feita esta validação, bem como as grandes forças e potenciais obstáculos à sua implementação;

Finalmente, no capítulo 8, apresentam-se as principais conclusões, discutem-se pontos de melhoria, sumariam-se as contribuições e equaciona-se trabalho futuro.

Parte I

Revisão Bibliográfica

2

Regulamentação e Normas para o Desenvolvimento de Dispositivos Médicos na UE

Até aos anos noventa, cada estado membro da União Europeia implementava a sua própria abordagem para regular os dispositivos médicos. Contudo, a fim de promover o mercado interno da UE, bem como para regular um mercado cada vez mais complexo, o Conselho Europeu introduziu novos regulamentos que ficaram conhecidas como «diretivas da nova abordagem». Estas definiam um conjunto de requisitos essenciais para assegurar a segurança e desempenho dos dispositivos [19]. Estas diretivas aplicam-se a todos os estados membros. Sendo assim, se um dispositivo cumprir com as normas estipuladas e receber o aval da UE num país, encontra-se certificado para todos os países [11, 19].

2.1 Um *framework* para Dispositivos Médicos

Esta estrutura montada pela UE para a promoção de um mercado único de circulação de dispositivos médicos é o que se chama de *Medical Device Framework* (MDF) e é discutivelmente uma das iniciativas mais

importantes de regulamentação legal aplicáveis a área da saúde [13]. Até ao ano de 2017, esta estrutura é composta por três diferentes diretivas para dispositivos médicos (DDM):

1. A Diretiva do Conselho 90/385/EEC relativamente a dispositivos médicos ativos implantáveis [20];

2. A Diretiva do Conselho 93/42/EEC relativamente a dispositivos médicos [21]. Esta, particularmente relevante no contexto deste documento;

3. A Diretiva 98/79/EC do Parlamento Europeu e do Conselho relativamente a dispositivos médicos de diagnóstico *in vitro* [22].

Estas três diretivas do MDF foram alteradas pela emenda 2007/47/EC [15]. Pretendiam fundamentalmente uniformizar os requisitos básicos de segurança e especificar um número de procedimentos relativamente a documentação e processos de auditoria que têm de ser respeitados para demonstrar que estes requisitos estão a ser respeitados. É obrigatório provar a conformidade com estes procedimentos para obter a certificação do Conselho Europeu (Artigo 3. [21]).

A aprovação dos dispositivos médicos na União Europeia é supervisionada por aquilo que é nomeado de «autoridade competente», tal como a *Medicines and Healthcare Products Regulatory Agency* no Reino Unido e o INFARMED [23] em Portugal [24]. Os dispositivos de baixo risco são declarados diretamente à autoridade competente, que poderá realizar inspeções e confirmar os padrões de produção e desenvolvimento, assim como rever a documentação técnica obrigatória. Para dispositivos mais complexos, a aprovação é levada a cabo por empresas independentes que se especializam na matéria e são designadas pelas autoridades competentes [13, 24].

O grau de risco associado à utilização do dispositivo médico é, então, o fator determinante para a classificação do mesmo. Esta classificação, por sua vez, ditará os requisitos de evidência necessários para uma autorização regulatória. Os dispositivos considerados de baixo risco são classificados como de Classe I, de médio risco como Classe IIa ou IIb e os de mais elevado risco como de Classe III. A tabela 2.1 faz um sumário dos critérios desta classificação.

A decisão de ser *compliant* com o conjunto de requisitos definidos na MDF representa um compromisso bastante caro para os produtores de dispositivos médicos, em termos de recursos de tempo, pessoal e financeiros. No entanto, estes submetem-se a este processo uma vez que pretendem vender os seus produtos a instituições médicas ou até mesmo a pacientes. De facto, o fardo que é o cumprimento destas normas é o preço necessário para garantir o «carimbo» do conselho europeu no dispositivo médico, que por sua vez garante o acesso a mercados lucrativos [13].

Tabela 2.1: Classificação dos dispositivos médicos pelo MDR de acordo com o seu grau de risco [19].

Classificação	**Descrição**	**Requisitos para o Mercado**	**Tempo estimado até aprovação**
Classe I	Dispositivos tipicamente simples em funcionamento e desenho. Apresentam risco negligenciável para a saúde humana. Exemplos incluem termómetros e luvas cirúrgicas.	É necessário declarar conformidade com os requisitos gerais	Não é necessária aprovação
Classe IIa	Os dispositivos classificados neste classe por norma apresentam baixo risco para a saúde humana. Alguns exemplos incluem catéteres digestivos, bombas de infusão e cadeiras-de-rodas elétricas.	Os produtores são obrigados a submeter um portfólio com literatura relevante e provar segurança e desempenho.	Cerca de 1 a 3 meses, mais o tempo necessário para endereçar eventuais inconformidades.
Classe IIb	Dispositivos cuja tecnologia apresenta relativamente alto risco para a saúde humana, como auxiliares de respiração e implantes ortopédicos.		
Classe III	Estes dispositivos implicam objetos cirúrgicos invasivos e com funcionamento a longo prazo, que podem colocar em risco a vida do paciente. Exemplos incluem *stents* coronários, *pacemakers* e desfribilhadores	Normalmente são necessários estudos clínicos. Os requisitos variam consoante o órgão aprovador.	Sem estimativa

Uma vez que o processo para adquirir certificação acarreta tantos encargos para as organizações, a MDF inclui o conceito de «finalidade prevista» na sua definição de dispositivo médico (Artigo 2º em [16]).

> *««Dispositivo médico», qualquer instrumento, aparelho, equipamento, software, implante, reagente, material ou outro artigo, destinado pelo fabricante a ser utilizado, isolada ou conjuntamente, em seres humanos, para um ou mais dos seguintes fins médicos específicos (...)»*

De acordo com o mesmo documento, entenda-se «finalidade prevista» como (Artigo 2º em [16]):

> *««Finalidade prevista», a utilização a que um dispositivo se destina, de acordo com as indicações fornecidas pelo fabricante no rótulo, nas instruções de utilização ou em materiais ou declarações promocionais ou de venda, e tal como especificada pelo fabricante na avaliação clínica;»*

Este simples conceito significa que sem a intenção explícita do produtor em criar um dispositivo médico, este não existe, ainda que aparentemente cumpra todos os restantes critérios.

2.2 *Standards* internacionais para o desenvolvimento de *software*: a IEC 62304:2006

No caso de o produtor do dispositivo assumir o compromisso de certificação de acordo com os padrões europeus, este deve reger-se por um conjunto de padrões internacionais que forneçam as orientações necessárias para a adquirir [25, 26]. No domínio da saúde, a ISO 13485:2016 [27] dita os requisitos necessários para fins regulatórios no que diz respeito a sistemas de gestão da qualidade (SGQ). Este *standard* é baseado na ISO 9001:2015 [28] e pode ser considerado uma extensão da mesma. Assim sendo, as organizações devem usar esta norma para avaliar a sua capacidade de responder de forma eficaz às necessidades dos seus clientes, assim como para medir a qualidade dos seus processos, recursos e artefactos produzidos.

A ISO 13485:2016, porém, não possui requisitos específicos para o desenvolvimento de *software*. A IEC 62304:2006 [17], que deve ser usada conjuntamente, colmata esta falha, oferecendo os padrões necessários para o ciclo de vida de desenvolvimento de *software* no que diz respeito à segurança e manutenção de soluções computacionais aplicadas a dispositivos médicos. Como fundação, a IEC 62304:2006 parte do princípio que a organização desenvolve de acordo com um SGQ, mas não exige a certificação de acordo com a ISO 13485:2016. Desta forma, a IEC 62304:2006 pode ser considerada como um suplemento da ISO 13485:2016 específica para o processo desenvolvimento de *software*. Algo semelhante acontece com as normas ISO/IEC 90003:2014 [29] e ISO 9001:2015, na medida em que a primeira fornece diretrizes para a aplicação da segunda em *software*.

A norma IEC 62304:2006 é baseada na ISO/IEC 12207:1995 [18], que embora seja um *standard* bastante compreensivo sobre os processos do ciclo de vida de desenvolvimento, já foi descontinuado, em primeiro lugar pela mais extensiva ISO/IEC 12207:2008 e seguidamente pela publicação da ISO/IEC/IEEE

12207:2017, a versão mais recente. Além disso, a comunidade ISO e IEC já publicou outros documentos que fornecem detalhe adicional relativamente aos processos de desenvolvimento, como a ISO/IEC 15504-5:2012 [30].

Ainda assim, a IEC 62304:2006 é considerada uma norma crítica para os criadores de *software* aplicado a dispositivos médicos porque é o único *standard* que fornece recomendações para este fim específico. Aliás, é a norma recomendada pelo Fórum Internacional de Reguladores de Dispositivos Médicos (IMDRF) para atingir o cumprimento das normas europeias [31]. Esta norma providencia aos desenvolvedores orientações de implementação baseadas nas piores consequências possíveis da falha da solução. Adicionalmente, a norma inclui uma classificação da segurança do *software* aplicado aos dispositivos médicos, onde esta é determinada com base no mesmo critério:

- No caso do produto classificado com Classe A, não ocorre qualquer lesão ou dano na saúde do utilizador em caso de falha no funcionamento do produto;
- Na eventualidade da classificação de segurança ser Classe B, poderá acontecer algum tipo de lesão, mas esta não é grave nem coloca em perigo a vida do utilizador;
- Caso a classificação seja Classe C, o *software* é considerado de alto risco e pode provocar lesões sérias ou mesmo morte dos intervenientes.

Há uma correspondência entre esta classificação e aquela apresentada na tabela 2.1. No entanto, esta correspondência não é direta, porque enquanto a classificação fornecida no MDF se aplica ao dispositivo médico no seu global, a classificação incluída na IEC 62304:2006 apenas se aplica aos componentes de *software*, que poderão não ser todos os componentes do dispositivo médico, aliás, o dispositivo até poderá incluir vários componentes diferentes, cada um com a sua classificação de segurança. Porém, se o componente de software for responsável por funcionalidades críticas, esta correspondência faz-se sentir, uma vez que uma falha em um apenas implica a falha do dispositivo. Neste caso, uma classificação Classe A corresponderia a uma classificação Classe III. É seguro afirmar, em conclusão, que a classificação de segurança do *software* nunca poderá ser mais elevada do que a classificação de segurança do dispositivo [25].

Para além de definições importantes, tais como as classificações de segurança, a norma delinia, também, os principais processos que o processo de desenvolvimento deve incluir [17]:

1. **O processo de desenvolvimento** - De particular interesse para este documento. A norma obriga a criação de uma planificação de desenvolvimento, responsável por orquestrar as atividades do processo e apropriada à dimensão e à classificação de segurança do sistema. O modelo usado no ciclo de desenvolvimento deverá estar detalhado na planificação, que também deverá pormenorizar os

documentos entregáveis a estratégia de rastreabilidade entre requisitos de sistema, *software*, teste e medidas de gestão do risco. As restantes atividades recomendadas no documento são: análise de requisitos, desenho de arquitetura, desenho detalhado, implementação, integração, testes de sistema e entrega.

2. **O processo de manutenção** - Embora por vezes classificada como uma atividade, a norma considera a manutenção como um processo independente. O produtor deve desenvolver uma planificação de manutenção, onde atenção é dada ao *feedback* que o produto recebe após ser entregue. Deverão ser detalhadas estratégias para receber, documentar, avaliar, resolver e acompanhar este *feedback*.

3. **O processo de gestão do risco** - O produtor deverá implementar um sistema de gestão de risco e incluí-lo na planificação do processo de desenvolvimento. Para tal, deve ser seguida a norma ISO 14971:2007 [32] e a norma IEC/TR 80002-1:2009 [33] (que orienta a implementação da anterior no contexto do *software*). Deverão ser identificados os itens de *software* que poderão contribuir para situações adversas, para as classificações de Classe A, B e C.

4. **O processo de gestão de configurações** - Deve ser incluída na planificação de desenvolvimento informação sobre a gestão de configurações. Isto inclui todas as tarefas e ações neste âmbito, uma listagem dos itens que devem ser controlados, versionamento e rastreabilidade de todas as mudanças a cada item.

5. **O processo de resolução de problemas** - A norma força o produtor a criar um relatório para cada problema identificado no produto, que deve incluir um tipo, âmbito e criticalidade. Além disso, deverão ser preparadas estratégias para investigar e resolver o problema, assim como manter a sua rastreabilidade. Finalmente, deverão ser criadas atividades de teste a atualização da documentação quando problemas são encontrados.

Mesmo fazendo todas estas recomendações, é fundamental reforçar as liberdades que são dadas pela norma IEC 62304:2006. A norma não restringe uma estrutura organizacional para os produtores de *software* no que diz respeito a papéis e responsabilidades e também não faz exigências quanto ao conteúdo específico da documentação a ser produzida. Mas fundamentalmente, a norma dá liberdade à organização de escolher o modelo de ciclo de vida de desenvolvimento.

As liberdades e imposições impostas pelo grande conjunto de normas e padrões internacionais relativamente ao desenvolvimento de produtos de *software* aplicados a dispositivos médicos certificados é importante para compreender a discussão que se coloca a seguir.

2.3 Uma mudança de paradigma

A norma IEC 62304:2006 considera o *software* como um subsistema embebido num dispositivo médico e, como consequência, os requisitos de sistema ou produto não estão incluídos na norma. A norma que os pretende incluir é a IEC 60601-1 [34], relativa a equipamento elétrico. Contudo, quando as diretivas referentes a dispositivos médicos foram emendadas em 2007 [15], o *software* por si só começou a poder ser classificado oficialmente como um dispositivo médico. Esta emenda revelou uma lacuna nos *standards* internacionais, uma vez que não existia nenhuma publicação com recomendações relativamente ao funcionamento *standalone* de um produto de *software* como dispositivo médico [35]. Hoje em dia, a norma IEC 82304-1:2016 [36] endereça as preocupações de segurança para o *software* na área da saúde que não tenciona ser comercializado em conjunto com *hardware* dedicado (aplicam-se plataformas TI, aplicações para *tablet*, etc).

A diretiva 2007/47/EC definia dispositivo médico da seguinte forma [15]:

> «Dispositivo médico», qualquer instrumento, aparelho, equipamento, software, material ou outro artigo, destinado pelo fabricante a ser utilizado, isolada ou conjuntamente, em conjunto com quaisquer acessórios incluindo software destinado pelo seu fabricante para ser usado especificamente para fins de diagnóstico e/ou terapêuticos e necessário para a sua correta aplicação, em seres humanos, para um ou mais dos seguintes fins médicos específicos:
>
> - diagnóstico, prevenção, monitorização, tratamento ou atenuação de uma doença;
> - diagnóstico, monitorização, tratamento, atenuação ou compensação de uma lesão ou de uma deficiência;
> - estudo, substituição ou alteração da anatomia ou de um processo ou estado fisiológico;
> - controlo de conceção.
>
> e cujo principal efeito pretendido no corpo humano não seja alcançado por meios farmacológicos, imunológicos ou metabólicos, embora a sua função possa ser apoiada por esses meios.

A classificação do *software* como um dispositivo médico de direito é reforçada no Recital 6 da mesma diretiva [15]:

«É necessário clarificar que o software no seu próprio direito, quando especificamente destinado pelo fabricante para ser usado por uma ou mais das finalidades enumeradas na definição de dispositivos médicos, é um dispositivo médico. Software para finalidades genéricas quando usado num ecossistema de saúde não é um dispositivo médico.»

As mudanças introduzidas por esta diretiva deram asas à possibilidade das organizações produzirem *software* certificado como dispositivo médico, como é o caso do *FibriCheck* [37], uma aplicação para *smartphone* de monitorização de arritmias.

Já em Abril de 2017, foi adotado o Regulamento para Dispositivos Médicos [16] 2017/45/EC (RDM), publicado no jornal oficial da UE no dia 5 de Maio de 2017 e que veio a substituir as diretivas 90/385/EEC e 93/42/EEC, bem como as suas emendas. O período de transição para a implementação do novo regulamento é de três anos [38]. É importante esclarecer a diferença entre regulamento e diretiva. Enquanto que o primeiro é diretamente aplicável em todos os estados membros, o segundo necessita de uma implementação na lei local.

Embora semelhante a uma primeira leitura, o novo regulamento difere substancialmente das suas diretivas predecessoras em alguns aspetos, nomeadamente no tema em discussão. A primeira diferença reside na própria definição de dispositivo médico presente no Artigo 2º:

> «Dispositivo médico», qualquer instrumento, aparelho, equipamento, software, implante, reagente, material ou outro artigo, destinado pelo fabricante a ser utilizado, isolada ou conjuntamente, em seres humanos, para um ou mais dos seguintes fins médicos específicos:
>
> - diagnóstico, prevenção, monitorização, previsão, prognóstico, tratamento ou atenuação de uma doença;
> - diagnóstico, monitorização, tratamento, atenuação ou compensação de uma lesão ou de uma deficiência;
> - estudo, substituição ou alteração da anatomia ou de um processo ou estado fisiológico ou patológico;
> - fornecimento de informações por meio de exame *in vitro* de amostras provenientes do corpo humano, incluindo dádivas de órgãos, sangue e tecidos.
>
> e cujo principal efeito pretendido no corpo humano não seja alcançado por meios farmacológicos, imunológicos ou metabólicos, embora a sua função possa ser apoiada por esses meios.

De notar a expansão da definição de dispositivo médico para equipamentos que procuram fazer previsão ou diagnóstico de doenças. Em conformidade, uma aplicação que recolha dados de sensores e os utilize para medir o risco do utilizador contrair uma determinada doença, ou mesmo avaliar a probabilidade de piorar condições já existentes, poderá ser certificada como um dispositivo médico de acordo com o RDM, ainda que o fabricante tenha de declarar explicitamente a finalidade proposta, de acordo com as possibilidades oferecidas neste regulamento.

O regulamento explicita, adicionalmente, uma exceção para dispositivos cuja finalidade proposta esteja relacionada com áreas não clínicas, tais como de estilo de vida e bem-estar, sendo que, na declaração de finalidade proposta, o fabricante é obrigado a dar um esclarecimento inequívoco que elimine dúvidas de uma classificação arbitrária para contornar a classificação como dispositivo médico, o que, como já vimos, acarreta um grande conjunto de encargos e recursos.

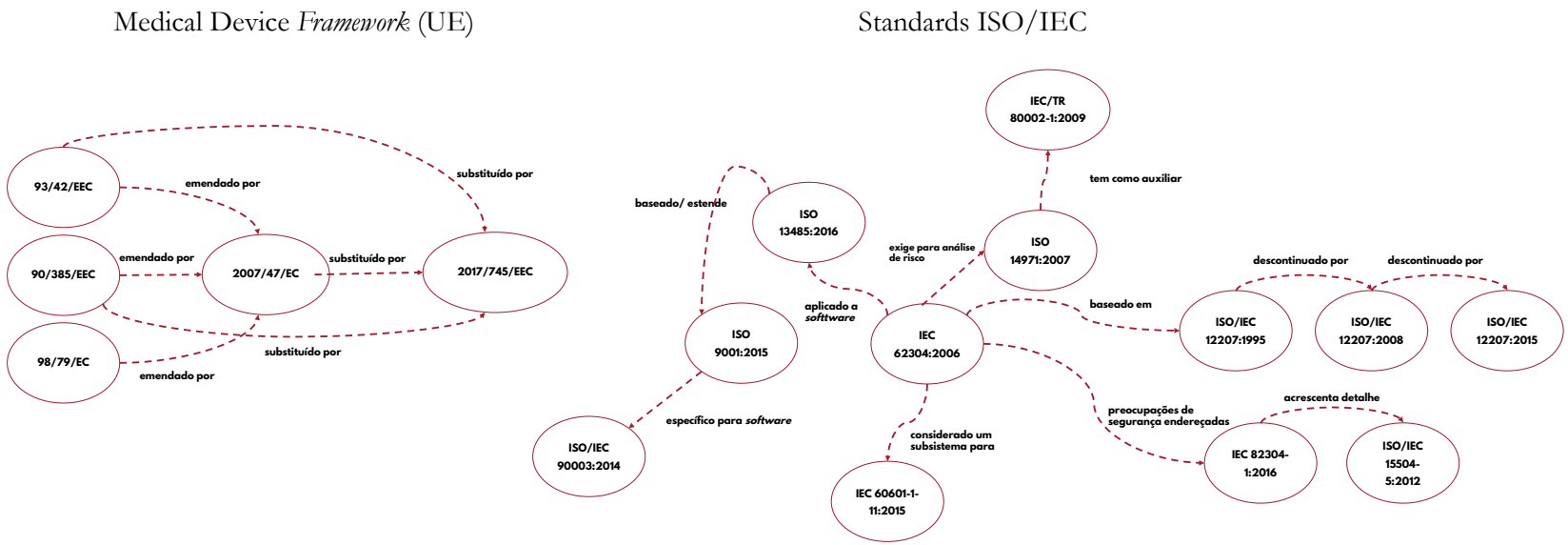
O RDM acrescenta, também, uma nova regra de classificação para software (Regra nº 11, anexo VIII):

> *«O software destinado a prestar informações utilizadas para a tomada de decisões com fins terapêuticos ou de diagnóstico é classificado na classe IIa, exceto se tais decisões tiverem um impacto que possa causar:*
>
> - *a morte ou uma deterioração irreversível do estado de saúde de uma pessoa, caso em que é classificado na classe III, ou*
> - *uma deterioração grave do estado de saúde de uma pessoa ou uma intervenção cirúrgica, caso em que é classificado na classe IIb.*
>
> *O software destinado a monitorizar os processos fisiológicos é classificado na classe IIa, exceto se se destinar à monitorização de parâmetros fisiológicos vitais, quando a natureza das variações desses parâmetros seja passível de resultar em perigo imediato para o doente, caso em que é classificado na classe IIb.*
>
> *Todo o restante software é classificado na classe I.»*

O alargamento da definição de finalidade médica em combinação com a nova regra de classificação adicionada tem implicações entusiasmantes para a indústria do *software*. Enquanto que a diretiva que já sobrevivia desde 2007 se focava no diagnóstico, monitorização e tratamento de doenças, no mais recente regulamento, para cair no âmbito da definição de dispositivo médico, a finalidade proposta pode ser de prognóstico ou previsão. Isto significa que muitos tipos de aplicação que não se enquadrariam no passado, passam a poder procurar certificação, provavelmente com uma classificação de IIa. Qualquer entusiasta da indústria do *software* consegue imaginar as implicações para organizações que trabalhem com *big data* e análise de dados, tecnologias com currículo comprovado em estratégias de prognóstico e previsão [39].

Em jeito de conclusão, é importante reforçar que o recital 19 do MDR salvaguarda que aplicações cuja finalidade proposta esteja relacionada com estilo de vida ou bem estar constituem uma exceção, na medida em que não serão classificadas como dispositivos médicos. Porém, toda a comunidade da engenharia de *software* deverá reconhecer que o caminho de uma aplicação de *lifestyle* para um dispositivo médico de Classe IIa ficou certamente mais curto.

Ainda assim, todos os aspetos discutidos neste capítulo podem representar uma carga difícil de lidar para qualquer empresa, até para as mais experientes. É um número considerável de normas e regulamentos, cada um deles com um nível de detalhe assinalável. Em termos de síntese, a figura 2.1 apresenta uma visão esquematizada de todo o ecossistema de normas relevantes.



93/42/EEC	Diretiva do Conselho Europeu relativamente a dispositivos médicos
90/385/EEC	Diretiva do Conselho Europeu relativamente a dispositivos médicos ativos implantáveis
98/79/EC	Diretiva do Parlamento Europeu e do Conselho relativamente a dispositivos médicos de diagnóstico *in vitro*
2007/47/EC	Diretiva do Parlamento Europeu e do Conselho que altera a diretiva 90/385/EEC e a diretiva 93/42/EEC
2017/745/EEC	Regulamento (UE) do Parlamento Europeu e do Conselho relativo aos dispositivos médicos que revoga as Diretivas 90/385/EEC e 93/42/EEC
IEC 62304:2006	*Software* de dispositivos médicos -- Processos do ciclo de vida do *software*
ISO 13485:2016	Dispositivos médicos -- Sistemas de gestão da qualidade -- Requisitos para fins regulatórios
ISO 9001:2015	Sistemas de gestão da qualidade -- Requisitos
ISO/IEC 90003:2014	Engenharia de *software*-- Diretrizes para a aplicação da ISO 9001:2008 a *software*
IEC 60601-1-11:2015	Equipamento médico elétrico -- Parte 1-11: Requisitos gerais para segurança básica e performance essencial
IEC 82304-1:2016	*Software* para a saúde – Parte 1: Requisitos gerais para segurança do produto
ISO/IEC 15504-5:2012	Tecnologia da informação – Avaliação do processo – Parte 5: Um modelo de avaliação exemplar para o ciclo de vida.
ISO/IEC 12207:1995	
ISO/IEC 12207:2008	Tecnologia da informação -- Processos do ciclo de vida do *software*
ISO/IEC 12207:2015	
ISO 14971:2007	Dispositivos médicos -- Aplicação da gestão do risco a dispositivos médicos
IEC/TR 80002-1:2009	Software para dispositivos médicos -- Parte 1: Orientação para a aplicação da ISO 14971 a *software* para dispositivos médicos

Figura 2.1: Ecossistema legal e *standards* internacionais relevantes para o processo de desenvolvimento aplicado a dispositivos médicos.

2.4 Enquadramento com a realidade portuguesa

Em Portugal, à semelhança dos restantes estados membros da União Europeia, o quadro legislativo referente a dispositivos médicos assenta na filosofia já mencionada neste documento, denominada de «Nova Abordagem», baseada na harmonização técnica e normativa que visa garantir níveis elevados de segurança e proteção da saúde dos utentes. A autoridade competente nomeada pelo governo português é o INFARMED - Autoridade Nacional do Medicamento e Produtos de Saúde, I. P [40].

Os dispositivos médicos, com exceção dos feitos por medida e dos destinados à investigação clínica, só poderão ser colocados no Mercado Europeu se apresentarem aposta a marcação CE como prova da sua conformidade com os requisitos essenciais que lhe são aplicáveis. De acordo com o INFARMED [41]:

> *«A marcação CE é um pré-requisito para a colocação no mercado e para a livre circulação dos dispositivos médicos no Mercado Europeu. Esta marcação tem um grafismo próprio e deve estar aposta pelo Fabricante de forma legível, visível e indelével.»*

As diretivas que fazem parte da MDF estão transpostas para a lei portuguesa a partir do Decreto-Lei n.º 145/2009 [42], publicado em Diário da República no dia 17 de junho de 2009. De acordo com este documento, tal como já foi abordado anteriormente, a avaliação de conformidade do dispositivo depende da sua classificação. No caso dos dispositivos de Classe I, a avaliação é feita pelo próprio fabricante, que apenas necessita de elaborar uma declaração e notificar a autoridade competente. Para dispositivos de uma classe superior, a avaliação é feita por um órgão notificado independente, nomeado pelo INFARMED.

De acordo com a nova realidade detalhada na secção anterior, há agora, ao abrigo da lei europeia, a possibilidade de um produto de *software* como uma aplicação para computador ou telemóvel obter a marca certificada do Conselho Europeu como dispositivo médico, que constitui uma garantia de conformidade com os requisitos essenciais de segurança e permite a livre circulação do dispositivo no mercado europeu. Assim sendo, o dispositivo poderá ser prescrito em Portugal por qualquer médico qualificado através dos métodos de prescrição eletrónica (PEM).

Além disso, não existe nada que impeça que este dispositivo, constituído apenas por *softwrae*, seja comparticipado pelo estado português. De acordo com o Decreto-Lei 97/2015 [43], publicado em Diário da República no dia 1 de junho de 2015, na secção II, no Artigo 23º:

> *1 — Quando se verifiquem razões de saúde pública ou vantagens económicas comprovadas, o Estado pode comparticipar, nos termos do presente decreto -lei, a aquisição de dispositivos médicos aos beneficiários do SNS e de outros subsistemas públicos de saúde, mediante requerimento do fabricante ou do seu representante com poderes para o efeito.*
>
> *2 — A competência para decidir a comparticipação dos dispositivos médicos referidos no número anterior ou, nos casos em que isso seja considerado adequado, a autorização de celebração*

de contrato de comparticipação, cabe ao membro do Governo responsável pela área da saúde, podendo ser delegada no conselho diretivo do INFARMED, I. P

Geram-se assim possibilidades que devem entusiasmar, tanto a comunidade da engenharia de *software*, como a comunidade médica em Portugal e nos restantes estados membros da União Europeia. É possível que os utentes possam beneficiar de ajuda financeira para poder utilizar *software* que lhes permita aumentar a sua qualidade de vida, evitar comportamentos de risco e usufruir das inúmeras potencialidades que as aplicações computacionais podem oferecer. É uma oportunidade para os profissionais de saúde aumentarem o seu grau de envolvimento no ecossistema eletrónico e informático que os rodeia e ajudar a combater as dificuldades enumeradas no capítulo 1.1, mas não só. É também uma oportunidade para a comunidade tecnológica associada à área da saúde sair das rotinas e repensar os seus processos e modelos de desenvolvimento face às novas realidades. É neste aspeto que se foca o próximo capítulo.

The foundation for software engineering is the process layer. The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software [44].

Roger S. Pressman

3

Paradigmas e Modelos de Desenvolvimento de Software

O IEEE fornece uma definição compreensiva do termo *Engenharia de Software* quando declara:

> *«the application of a systematic, disciplined, quantifyable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.»* [45]

Já Sommerville, define o mesmo termo da seguinte forma:

> *«Software engineering is an engineering discipline whose focus is the cost-effective development of high-quality software systems.»* [46]

Qualquer definição será o ponto de partida para uma discussão sobre um aspeto: **qualidade**. Neste caso, qualidade do *software* desenvolvido.

A engenharia de *software* funciona por camadas (Figura 3.1) e, tal como qualquer disciplina da engenharia, apoia-se num compromisso organizacional com a qualidade que deverá fomentar uma cultura de melhoria contínua. Sendo assim, a base principal na qual assenta a engenharia de *software* é a camada processual [44]. O processo de engenharia fornece uma interface para que as camadas tecnológicas possam comunicar, permitindo um desenvolvimento eficiente de produtos computacionais.

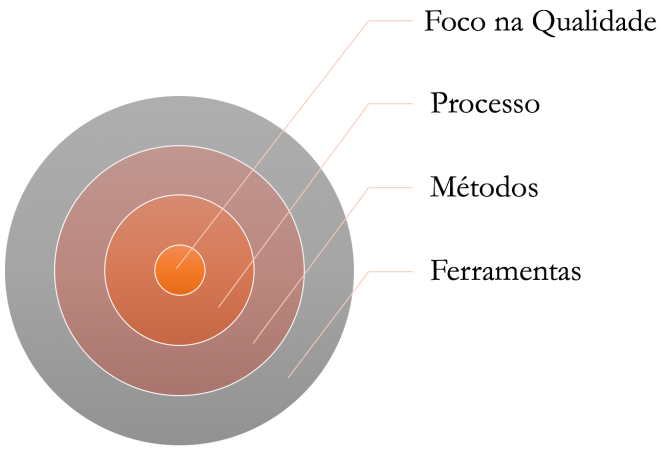


Figura 3.1: As camadas da Engenharia de *Software*. Adaptado de [44]

Contudo, uma abordagem «sistemática, disciplinada e quantificável» aplicada a uma equipa de desenvolvimento poderá facilmente ser desenquadrada de outra. A disciplina é necessária, mas também não deve ser impeditiva da adaptabilidade [44]. A bibliografia é clara no sentido de que não existe uma abordagem ideal e generalizada para criar produtos de *software*. A vasta panóplia de sistemas e organizações existentes justificam a necessidade de uma grande diversidade de abordagens para o processo de desenvolvimento. Ainda assim, noções básicas de processos de engenharia servem de pilar para todas estas técnicas e são estas que constituem a essência da engenharia de *software* [46].

3.1 O Processo de Engenharia de *Software*

Neste contexto, PROCESSO define a estrutura que deve ser estabelecida de forma a entregar tecnologia de forma eficiente. O processo de engenharia de *software* forma uma base para o controlo e gestão de projetos e fornece o contexto a partir do qual os métodos técnicos são aplicados, os produtos finais (modelos,

artefactos, documentos) são criados, as *milestones* [*] definidas, a qualidade é assegurada e a mudança gerida [44].

A definição de processo não é consensual na bibliografia, contudo, pode ser inferido que representa um conjunto de atividades, ações e tarefas que são desempenhadas quando algum produto é criado [44, 47].

- Uma atividade tem a finalidade de atingir um objetivo genérico, como por exemplo, estabelecer um veículo de comunicação com os *stakeholders* [†] e é aplicável independentemente do domínio aplicacional, dimensão ou complexidade do projeto.
- Uma ação, como o desenho da arquitetura da solução, engloba um conjunto de tarefas que produzem um produto final de trabalho. Neste caso, o produto final seria um modelo da arquitetura.
- Finalmente, a tarefa é a mais atómica das três. Esta foca-se num objetivo pequeno e bem delineado (como por exemplo programar um teste unitário) e produz um resultado tangível.

É fundamental reforçar que um processo não pretende ser uma prescrição rígida de como construir produtos de *software*. Em detrimento, deve ser encarado como uma abordagem adaptável que permite à equipa escolher o conjunto de tarefas e ações mais apropriadas, tendo em conta os recursos disponíveis. O objetivo deverá sempre ser a entrega atempada de *software* com um nível de qualidade que satisfaça aqueles que patrocinaram o seu desenvolvimento.

Ainda assim, é possível chegar a um pequeno conjunto de atividades que são comuns a todos os modelos de desenvolvimento:

1. **Planeamento** [44] - Esta atividade é ignorada por alguns autores, potencialmente por considerarem que precede o processo de desenvolvimento. Não obstante, é importante construir um mapeamento de todas as tarefas, técnicas ou não, identificar os riscos existentes, medir os recursos disponíveis e esclarecer a agenda tanto do projeto como da equipa.
2. **Especificação** [46] - No decorrer desta atividade, engenheiros e *stakeholders* criam a definição do produto que se pretende que seja desenvolvido e esclarecem potenciais restrições e impedimentos.
3. **Modelação** [44] ou Desenho - Consiste na criação de abstrações que descrevam a solução a desenvolver. Identificam-se fluxos aplicacionais, discutem-se padrões de arquitetura e selecionam-se as tecnologias e convenções pertinentes.

[*] Em gestão de projetos, o termo *milestone* é utilizado para marcar a conclusão de um conjunto de tarefas ou fases e serve de âncora na cronologia do projeto. Não é uma atividade nem possui duração.

[†] Um indivíduo, grupo, ou organização que poder afetar, ser afetado ou influenciar de alguma forma uma decisão, atividade ou desenrolar de um projeto.

4. **Construção** [44] ou Desenvolvimento [46] - Atividade onde se codifica a solução através de técnicas de programação e ciência da computação.

5. **Validação** [46] - Apesar de haver autores que colocam esta atividade em conjunto com a etapa de codificação [44], decidimos destacar esta atividade devido à sua importância, nomeadamente no setor dos sistemas críticos. No decorrer da validação, o produto desenvolvido é contraposto aos requisitos especificados de forma a garantir a qualidade da solução.

6. **Manutenção e Evolução** [46] - O *software* deve ser capaz de evoluir de acordo com potenciais alterações exigidas pelos *stakeholders*.

3.2 Modelos do Ciclo de Vida de Desenvolvimento

Modelos para o ciclo de vida de desenvolvimento fornecem um guião de importância fulcral para o sucesso de qualquer projeto. Na verdade, é sempre discutível qual será o modelo indicado a seguir: aliás, talvez a única escolha imprudente seja aquela de uniformizar em absoluto e impor o mesmo modelo de desenvolvimento para todas as situações relacionadas com o desenvolvimento de *software* [12].

Os benefícios de seguir um modelo, são, porém, consensuais. Sem possuir etapas de desenvolvimento, torna-se difícil de saber o estado do projeto como um todo, gerir *milestones* e atribuir tarefas aos elementos da equipa. Este controlo permite não só aumentar a qualidade do produto resultante, como também tornar o projeto economicamente viável, através da definição de orçamentos e agendas. Além disso, no contexto de sistemas críticos, este controlo assume uma magnitude ainda maior uma vez que as autoridades reguladoras necessitam de comprovar a qualidade do processo de engenharia.

A natureza intangível do *software* permite a existência de uma grande variedade de ciclos de vida de desenvolvimento de *software* (CVDS). Estes modelos podem variar em diversos aspetos [44]:

- no fluxo geral que coordena atividades, ações e tarefas, assim como as dependência entre elas;
- nos produtos do trabalho que são exigidos e criados;
- na forma como as atividades de gestão de qualidade são aplicadas;
- no modo como as atividades de controlo e rastreabilidade do projeto são aplicadas;
- no nível geral de rigor e detalhe na descrição do projeto;
- no nível de envolvência dos *stakeholders* com o projeto;
- no grau de autonomia dado à equipa de desenvolvimento;

- no grau de organização da equipa no que diz respeito aos papeis atribuídos a cada elemento.

Apesar disto, todos os modelos processuais deverão ser capazes de acomodar as atividades genéricas discutidas anteriormente.

3.2.1 Modelo *Waterfall*

O modelo em cascata, ou *waterfall*, é considerado o modelo documentado mais antigo para o ciclo de desenvolvimento [46], e é creditado a Winston W. Royce [48] que o apresentou num artigo publicado em 1970. Apesar de Royce ter descrito o modelo, o nome *waterfall* apenas apareceu numa publicação formal no ano de 1976 [49], nome que pretende ilustrar a comunicação unidirecional entre as atividades principais do modelo.

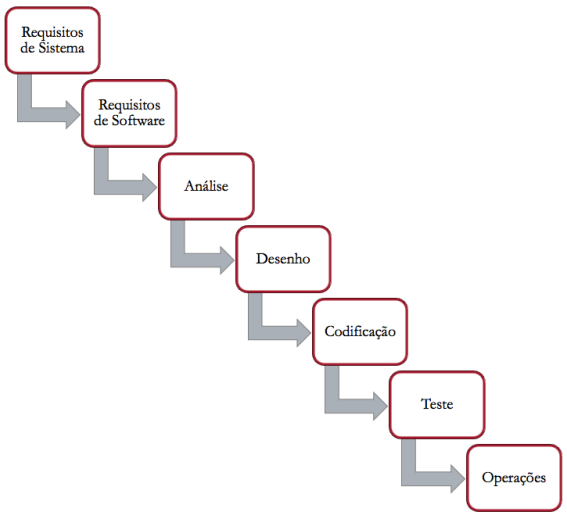


Figura 3.2: Modelo *Waterfall*. Adaptado de [48]

Este modelo é linear, na medida em que sugere uma abordagem sistemática e sequencial, tal como ilustrado na figura 3.2. No modelo original, era previsto um certo grau de iteração entre atividades consecutivas, através daquilo a que o autor apelidou de *feedback loops*, contudo, a grande maioria das organizações encara o processo como estritamente unidirecional.

O processo inicia-se com uma atividade de levantamento e especificação de requisitos, onde, em conjunto com o cliente, são identificados os serviços a fornecer pela solução final, assim como o conjunto de restrições que afetam o mesmo. De seguida, inicia-se uma atividade de desenho do sistema, onde os requisitos são usados para construir uma arquitetura global, descritas as abstrações do sistema, assim como os seus componentes e interações entre eles. Segue-se a etapa de implementação, onde é concretizado o desenho numa solução de *software*. Esta solução é, seguidamente, integrada e testada e só depois entregue ao cliente. É ainda prevista uma atividade de manutenção e operações, que normalmente implica a correção de erros, melhoramento da implementação do sistema e evolução do mesmo se novos requisitos forem identificados [46, 48].

A cada fase do processo é dada prioridade à produção de documentação compreensiva das tarefas e ações desempenhadas.

Uma das principais adaptações (na realidade, uma interpretação alternativa) do modelo *waterfall* é o chamado modelo em *V*, ou *V-Model* [50], ilustrado na figura 3.3. A representação em *V* do modelo linear

clássico fornece uma vista tridimensional das atividades técnicas do ciclo de vida de desenvolvimento. No lado esquerdo do gráfico, realizam-se etapas de decomposição e definição do sistema de forma sequencial e com um nível de detalhe crescente; contudo, no decorrer do fluxo ascendente do lado direito, são realizadas atividades de integração e validação num nível crescente de abstração, culminando ao nível do sistema [50]. Uma particularidade interessante incluída em muitas implementações do modelo em *V* é o planeamento das atividades de teste paralelamente à respetiva atividade de desenho ou desenvolvimento [51].

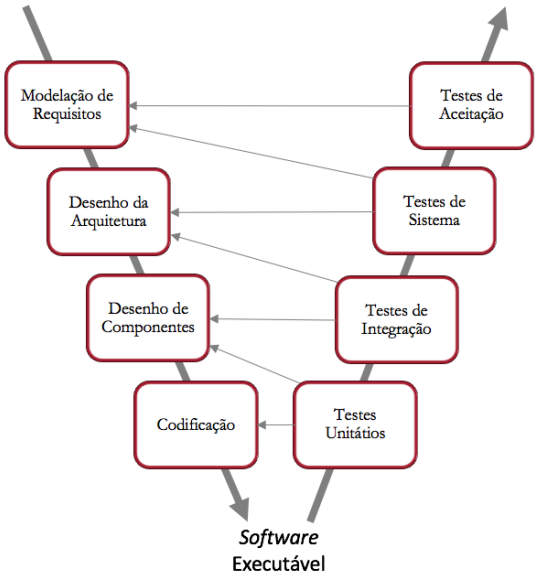


Figura 3.3: O *V-Model*. Adaptado de [44]

Na realidade, não existe nenhuma diferença real entre o Modelo em *V* e o modelo clássico em cascata. O primeiro fornece, sim, uma interpretação do segundo que permite visualizar claramente como as atividades de *quality assurance* [‡] se relacionam com o trabalho de engenharia realizado em atividades anteriores do ciclo [44].

O modelo em cascata e as suas adaptações continuam a ser os preferidos no desenvolvimento para a indústria médica, devido à sua natureza orientada à planificação (*plan-driven)* [52]. O rigor, atenção à qualidade e prioridade dados à produção de documentação estimulam a criação de artefactos tangíveis, necessários para satisfazer as entidades reguladoras. A tabela 3.1 faz uma síntese das principais vantagens e desvantagens do ciclo em cascata.

O *V-Model* é considerado particularmente interessante no caso do desenvolvimento para dispositivos médicos, uma vez que força o planeamento de diferentes atividades de teste em paralelo com as etapas de desenvolvimento. O peso dado por este modelo às diferentes etapas de validação tem como consequência a produção de evidências detalhadas da qualidade do processo.

3.2.2 Modelos Iterativos e Incrementais

O *software* moderno é muitas vezes caracterizado por mudanças contínuas, agendas curtas e por uma necessidade incontornável de satisfazer o cliente, ou utilizador. Na maioria dos casos, o *time-to-market* é

[‡]Traduzido literalmente para «garantia de qualidade». *Quality assurance*, ou simplesmente QA inclui atividades administrativas e procedimentais que são implementadas num sistema de qualidade a fim de garantir cumprimento com os requisitos e objetivos de um produto, serviço ou projeto.

Tabela 3.1: Vantagens e desvantagens do ciclo *Waterfall*.

Prós	Contras
Intuitivo e de fácil compreensão;	Não é considerado realista para software com um grau elevado de complexidade, na medida em que a reação à mudança não é direta e pode provocar confusão;
Bem documentado e conhecido na indústria;	A primeira versão do produto apenas é finalizada em etapas finais do ciclo;
Força um grau avançado de compreensão dos sistema antes da implementação;	Os primeiros testes são realizados em etapas avançadas do ciclo;
Milestones e pontos de controlo claramente definidos;	Podem ser gerados *bottlenecks*, quando elementos da equipa são obrigados a aguardar pela conclusão das tarefas de outros, das quais estão dependentes.
Enquadra-se com outros modelos processuais de engenharia, para além do *software*;	
Forte documentação.	

a principal prioridade da equipa de gestão [44]. Estas são questões que não são respondidas pelos modelos lineares, e justificam a procura por modelos iterativos e incrementais.

O conceito de iteratividade no desenvolvimento de *software* não é algo recente [53]. Na verdade, a publicação de Royce [48], que deu origem ao pensamento em cascata, já falava de pequenos ciclos incrementais. O conceito implica desenvolvimento em ciclos repetitivos. Já desenvolvimento incremental implica produzir e entregar *software* em incrementos, em vez de numa única instalação. Cada iteração do processo produz um novo incremento [46].

Quando é usado um ciclo de vida incremental, o primeiro incremento é, por norma, um produto central para a funcionalidade global da solução. Os requisitos mais básicos são endereçados, mas muitas *features* [§] suplementares permanecem por entregar, aliás, algumas das funcionalidades podem até nem ser conhecidas. O produto *core* pode ser utilizado e avaliado após a entrega e o resultado desta avaliação fornece um contributo para o planeamento das iterações seguintes. Este plano deve endereçar a modificação do produto instalado de forma a satisfazer as necessidades do cliente e o processo é repetido após cada entrega até que o produto final esteja concluído [44, 46].

Os modelos iterativos e incrementais estão também intimamente ligados com os conceitos evolucionários, que implicam a entrega iterativa de um produto cada vez mais completo. Entre as opções evolucionárias para o desenvolvimento de *software*, destacam-se duas: Prototipagem e Modelo em Espiral.

[§] De acordo com o IEEE, «Uma característica distintiva de um item de software (tal como performance, portabilidade ou funcionalidade)».

- **Prototipagem** - Um protótipo é um exemplar, um modelo ou uma versão do produto a ser desenvolvido, que visa testar um conceito. No contexto evolucional, e em contraste com protótipos descartáveis (*throwaway*), o protótipo é desenvolvido com atenção à qualidade e passa por atividades de especificação e teste [54]. Apenas implementa requisitos confirmados e é utilizado como estratégia para levantamento de requisitos adicionais. Iterações ocorrem na medida em que o protótipo é evoluído com o objetivo de satisfazer as necessidades dos *stakeholders*, além de clarificar para todos o entendimento global da solução.

- **Modelo em Espiral** - Proposto originalmente por Boehm [55], o modelo espiral é um modelo evolucionário que propõe uma combinação da natureza iterativa da prototipagem com a sistematicidade do *waterfall*. Imaginando uma espiral, o processo evolucionário começa quando a equipa de desenvolvimento realiza uma série de atividades em circuito à volta da mesma, iniciando no centro. Em cada iteração à volta do circuito, estas atividades são desempenhadas (planificação, modelação, construção, teste), cada vez com um grau de sofisticação e de detalhe maior. Ao final de cada ciclo, é potencialmente construído um protótipo, é recolhido *feedback* do cliente, é revista a planificação e os riscos são medidos. De acordo com o autor, este modelo prima por apresentar uma abordagem cíclica para o crescimento incremental do sistema em termos de definição e implementação, enquanto que o grau de risco diminui. Além disso, são definidas uma séria de *milestones* estratégicas para assegurar o nível de compromisso dos *stakeholders* [55].

À semelhança de outros paradigmas, os modelos iterativos não são uma panaceia universal. Poderá, nomeadamente, ser difícil de convencer os clientes que a abordagem evolucionária é controlável. A tabela 3.2 faz um apanhado dos principais prós e contras referentes aos ciclos de vida iterativos e incrementais.

3.2.3 Modelos Ágeis

Nos anos 80, assim como no início da década de 90, o ponto de vista generalizado na comunidade do *software* era que a melhor abordagem para atingir a qualidade consistia num planeamento criterioso, *quality assurance* e, de uma forma global, num processo controlado e rigoroso de engenharia [46]. Esta visão provinha da experiência no desenvolvimento de sistemas críticos.

Contudo, quando esta abordagem pesada começou a ser aplicada a sistemas de negócio mais pequenos, o *overhead* envolvido tornou-se tão significativo, que dominava completamente o processo de desenvolvimento. Os recursos eram aplicados em grande maioria no cumprimento do processo, em detrimento das atividades de construção e teste.

A insatisfação com as abordagens *plan-driven* levou a que, em 2001, Kent Beck e outros 16 reconhecidos desenvolvedores de *software*, escritores e consultores (auto-apelidados de *Agile Alliance*) assinassem

Tabela 3.2: Vantagens e desvantagens dos modelos incrementais

Prós	Contras
Produção de *software* funcional num estágio precoce do ciclo de vida de desenvolvimento;	Cada iteração é rígida e não existe *overlap* entre iterações;
Mais flexível do que o modelo linear, na medida em que a reação à mudança implica menos custos;	Poderá haver problemas na definição da arquitetura e modelação do sistema, associado ao grau de indefinição nos requisitos;
A Gestão de risco é facilitada uma vez que é feita a cada iteração;	Poderá ser exigente ao nível de recursos uma vez que todas as atividades do ciclo de vida são realizadas com frequência e menos espaçadas temporalmente;
Milestones e pontos de controlo claramente definidos;	Difícil estimar uma data de conclusão para o projeto.
Os defeitos basilares são identificados precocemente;	
O *Feedback* frequente do cliente permite gerir o grau de satisfação do mesmo e diminuir o risco.	

o «Manifesto para Desenvolvimento de Software Ágil» [56], que declara:

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> - *Individuals and interactions over processes and tools*
> - *Working software over comprehensive documentation*
> - *Customer collaboration over contract negotiation*
> - *Responding to change over following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more.*

O aspeto basilar que torna as metodologias ágeis tão relevantes é a sua facilidade de reação à mudança. Num modelo tradicional, o custo da mudança aumenta de forma não linear à medida que o projeto pro-

gride [44]. Isto significa que é relativamente fácil de acomodar a mudança quando se levanta os requisitos iniciais, mas o mesmo não acontece quando o projeto se aproxima dos seus estágios finais. Defensores dos métodos ágeis defendem que um modelo que siga os princípios do manifesto deverá contrariar a curva de progressão de custo-tempo, permitindo que a equipa consiga lidar com a mudança facilmente em qualquer momento do ciclo de vida de desenvolvimento.

Mas a filosofia da agilidade é mais do que uma resposta eficiente à mudança. É um paradigma de trabalho que abriga as declarações presentes no manifesto. Encoraja-se a comunicação facilitada entre equipas e indivíduos, nomeadamente entre os membros da equipa de desenvolvimento, mas também entre profissionais, técnicos e outros relacionados com a vertente de negócio do projeto. Enfatiza-se a entrega rápida de soluções funcionais. Assume-se o cliente como um membro integrante da equipa de desenvolvimento, removendo barreiras de comunicação. Reconhece-se que a planificação tem limites e deve ser flexível [57].

Contrariamente, os modelos de desenvolvimento tradicionais e orientados à planificação falham principalmente porque não contabilizam os fatores humanos que dizem respeito às equipas envolvidas. Os engenheiros não devem ser vistos como robôs, uma vez que possuem diferenças significativas no seu nível técnico e comunicativo, mas também na sua forma de estar. Uma vez que a consistência é uma fraqueza humana, metodologias que exijam disciplina são frágeis [58]. A filosofia *agile* tenta contrariar esta tendência, introduzindo tolerância relativamente à equipa de engenharia.

Existem vários modelos concretos da filosofia ágil. Aqui destacam-se dois dos mais celebrados: Extreme Programming e SCRUM.

3.2.3.1 *Extreme Programming (XP)*

Criada por Kent Beck [59] , esta metodologia leva àquilo que foram consideradas as «melhores práticas» no desenvolvimento a níveis extremos. O XP engloba quatro atividades principais: planeamento, desenho, codificação e teste.

No decorrer do planeamento, realizam-se uma série de práticas de levantamento de requisitos onde se pretende que a equipa técnica adquira conhecimento de negócio em relação à solução. Estes requisitos manifestam-se sob a forma de cenários (*user stories*) e são mapeados diretamente para tarefas. O cliente é envolvido diretamente na construção e prioritização destas tarefas, assim como na especificação dos critérios e testes de aceitação [46, 59].

A fase de desenho providencia uma linha condutora para uma ação de implementação. O XP segue um pensamento KIS (*keep it simple*), o que significa que uma solução mais simples é privilegiada relativamente a uma solução mais complexa. Neste sentido, a metodologia encoraja a prática de *refactoring*, que consiste no procedimento de modificar o *software* de forma a que o seu comportamento funcional

não seja alterado, mas a sua estrutura interna seja melhorada. É importante reforçar que o desenho ocorre antes e depois da implementação, na medida em que o *refactoring* existe continuamente no decorrer da construção.

Os programadores trabalham em pares e desenvolvem testes antes de começarem a programação. Na medida em que os pares de programadores concluem as suas tarefas, vão integrando o seu trabalho com o dos restantes de uma forma contínua. Os testes deverão ser automatizados e executados com uma frequência diária.

O *extreme programming* representa uma mudança de paradigma radical relativamente àquilo que foi discutido nas secções anteriores. O processo começou a ser pensado sob a forma de valores (comunicação, *feedback*, simplicidade, coragem e respeito) e de princípios, tais como a integração contínua, desenho incremental e o *pair programming* em vez de ser pensado como um conjunto de atividades bem definidas e delineadas. Nas palavras de Beck [56]:

> *«Extreme Programming (XP) is about social change. It is about letting go of habits and patterns that were adaptive in the past, but now get in the way of us doing our best work. It is about giving up the defenses that protect us but interfere with our productivity. It may leave us feeling exposed.»*

3.2.3.2 SCRUM

O *SCRUM* é mais uma metodologia popular para o desenvolvimento de *software* de acordo com a ideologia ágil. O modelo foi introduzido por Ken Schwaber e Jeff Sutherland [60], que acreditavam que os processos usados na época, pela indústria, estavam «estragados» [61].

Este modelo partilha bastantes dos valores do XP, nomeadamente naquilo que diz respeito à procura de qualidade com entregas incrementais num processo iterativo. Contudo, introduz aspetos diferenciadores no que diz respeito à constituição das equipas e ao fluxo de desenvolvimento a ser seguido pelas mesmas [61].

A equipa num projeto *SCRUM* é constituída por um *Product Owner*, pela equipa de desenvolvimento e por um *SCRUM Master*. As equipas devem ser auto-organizadas e ter responsabilidades transversais a todo o projeto, o que as torna independentes. O *Product Owner* é a pessoa responsável por maximizar o valor da solução construída pela equipa de desenvolvimento. É ele que deve ordenar e prioritizar as tarefas de desenvolvimento (*user stories*) num repositório a que se chama *backlog* do produto. À equipa de desenvolvimento compete garantir uma entrega funcional de acordo com o planeado a cada incremento. A cada período de desenvolvimento entre entregas dá-se o nome de *sprint*. O *SCRUM Master* é o responsável por garantir que o processo de *SCRUM* se realiza e é entendido. Este fornece uma inter-

face entre os intervenientes, como o *Product Owner*, a equipa de desenvolvimento e a própria organização [62].

Tabela 3.3: Vantagens e contrapartidas dos modelos ágeis

Prós	Contras
Reação rápida à mudança;	O envolvimento dos *stakeholders* pode ser difícil dependendo da natureza do mesmo. Além disso, pode implicar recursos para a formação do cliente;
Encoraja o envolvimento entre *stakeholders* potenciando a transparência e a clareza nos requisitos;	Necessário um elevado nível de competências dos participantes, quer a nível técnico, quer a nível de competências de comunicação e maturidade uma vez que a equipa é auto-organizada;
Estimula a colaboração ativa entre os elementos da equipa, facilitando a descoberta de impedimentos e a partilha de conhecimento;	Reúne também as desvantagens dos modelos iterativos e incrementais previamente apresentadas.
Tratando-se de um processo com natureza iterativa, aglomera as vantagens dos restantes processos de natureza iterativa e incremental.	

Na sua essência, o movimento *agile* é um esforço para ultrapassar as fraquezas no processo convencional de engenharia de *software*. Contudo, é fundamental perceber que o reconhecimento dos desafios das realidades do desenvolvimento moderno não deverá ter como consequência o abandono dos princípios basilares descritos nos modelos tradicionais.

É também importante perceber que, apesar da defesa evangélica por parte dos promotores das metodologias ágeis, existem contrapartidas a ter em conta. Apesar da ideia do envolvimento por parte do cliente ser atraente, é extremamente dependente da eventualidade do mesmo apresentar disponibilidade para acompanhar a equipa de desenvolvimento, assim como conseguir representar o conjunto global de *stakeholders*. Além disso, é possível que elementos da equipa não se identifiquem com a natureza intensa de envolvimento que os métodos ágeis obrigam, o que pode prejudicar as interações no seio da equipa.

A prioritização de itens pode, também, ser problemática caso o conjunto de *stakeholders* seja grande e disperso em natureza. *Stakeholders* diferentes frequentemente atribuem prioridades diferentes face à mudança.

3.3 Equilíbrio da disciplina com a agilidade

A disciplina é o alicerce o sucesso de qualquer diligência. Os atletas treinam para ganhar medalhas, os músicos aperfeiçoam a sua técnica e os engenheiros aplicam processos. No mundo do *software*, a palavra

craft, que pode ser traduzida literalmente para «ofício», começa a ganhar popularidade para descrever a arte de desenvolver *software* sofisticado e com olho para o detalhe [63]. Este termo está associado a meticulosidade e rigor técnico. A disciplina proporciona força e conforto perante situações complicadas, cria organização e experiência [64].

Pelo contrário, a agilidade é o contrário da disciplina. A agilidade é inventora e flexível quando a disciplina é forte e previsível. A agilidade caracteriza-se pela capacidade de reagir à mudança e de adaptar a novos ambientes fora da zona do conforto.

Neste documento apresenta-se a forte convicção de que qualquer empreendimento exige disciplina e agilidade para triunfar. Isto é tão verdade no desenvolvimento de *software* como para qualquer outra atividade profissional como o desporto ou a arte. Em *Balancing Agility and Discipline* [64], Boehm sugere que, sem agilidade, a disciplina traduz-se em forte burocracia e estagnação mas, por outro lado, o cenário inverso não passa do entusiasmo de uma *startup* que ainda não começou a pensar em lucros mensuráveis. Empresas de sucesso devem colocar estas duas características nos pratos da balança e fazer os ajustes necessários, baseando-se nos seus objetivos, recursos e cultura.

Como vimos no capítulo 1, à semelhança do que acontece um pouco por todos os setores da indústria, a área da saúde está em mudança, na medida em que o *software* desempenha um papel cada vez mais influente [6, 7]. Além disso, e com as mudanças na regulamentação dos dispositivos médicos que foram apresentadas na secção 2.3, o *software* que é qualificado como dispositivo médico vai mudar radicalmente. Não serão só sistemas embebidos e o *software* de longa duração a serem classificados como dispositivos médicos, mas também qualquer tipo de sistema *standalone*. Destaca-se o crescimento das aplicações móveis neste setor, que originaram o termo *mHealth* [65, 66].

Até agora, a disciplina tem sido a estratégia preferida da comunidade para o desenvolvimento de *software* certificado aplicado a dispositivos médicos [52]. Estes métodos *plan-driven* concentram-se na qualidade dos artefactos e na previsibilidade do processo para reduzir o risco e tornar o desenvolvimento menos caótico. Contudo, o preço da disciplina é a falta de flexibilidade face à mudança e o custo elevado que esta representa, riscos difíceis de enfrentar quando falamos de aplicações mais voláteis como as de dispositivos móveis.

Nas últimas duas décadas, a comunidade do desenvolvimento de *software* tem sido desafiada pelo movimento *agile* a mudar radicalmente a sua perspetiva. A metodologia ágil encoraja programadores a despirem as cadeias «rígidas» dos seus processos e a abraçarem a mudança. Segundo esta abordagem, os modelos têm ciclos curtos e incrementais, alto nível de envolvimento com o cliente e pretendem adaptar em vez de ser previsíveis.

De facto, o número de organizações a adotar os métodos ágeis nos seus ciclos de desenvolvimento tem vindo a crescer [67, 68], dando origem a projetos com menos documentação focados na rápida satisfação do cliente. Ainda assim, muitas empresas mostram preocupações com os riscos envolvidos, nomeada-

mente na aptidão destas metodologias face a projetos de natureza *safety-critical* [69, 70].

Infelizmente, as duas abordagens são encaradas na maioria da literatura como adversárias e completamente contraditórias. Enquanto que os defensores da metodologia *agile* criticam os processos tradicionais por serem desumanos, do outro lado respondem com acusações de falta de qualidade e de rigor, já para não falar de um ambiente demasiado descontraído para estimular o profissionalismo. Nestas condições de quase «fanatismo», geram-se oportunidades para interpretar erradamente ambas as abordagens, até porque ambas continuam a evoluir no ecossistema tecnológico.

Neste documento, pretende-se manifestar uma opinião de que existem circunstâncias distintas em que cada uma das abordagens é favorecida e tem condições para potenciar melhor o projeto. Ainda assim, não há motivo que justifique o facto de não poderem ser combinadas num processo híbrido que tente recolher as melhores características de cada abordagem, mediante uma série de fatores.

Cada uma das metodologias tem uma «zona de conforto», sendo que, para as *plan-driven* geralmente são projetos de grande dimensão e complexidade, normalmente com componentes *safety critical* e outros atributos de alto rico. Regra geral os requisitos são estáveis e o âmbito bem definido. Os métodos ágeis estão mais confortáveis quando os sistemas e equipas de desenvolvimento são mais pequenos, os clientes ou utilizadores mostram disponibilidade para participar ativamente no projeto e os requisitos e ambiente são voláteis [64, 71]. A tabela 3.4 pretende fazer um sumário, em jeito de comparação, das condições que caracterizam cada uma das metodologias.

No início desta secção, começou-se por apresentar o argumento de que qualquer empreendimento requer disciplina e agilidade. No âmbito do setor da saúde, especificamente no desenvolvimento de *software* certificado e *compliant*, a disciplina tem sido privilegiada uma vez que que a natureza dos projetos se enquadravam com o panorama descrito na coluna da direita da tabela 3.4. Porém, com a entrada do *software* de prognóstico e previsão prevista pelos regulamentos mais recentes, de que forma mudará este enquadramento?

Daquilo que vemos pela análise da tabela 3.4, há lógica em assumir que apenas um dos fatores priveligia claramente as metodologias disciplinadas: o impacto dos erros. Relativamente às restantes, dependerão fortemente do âmbito do projeto e é possível imaginar múltiplos cenários em que uma metodologia ágil se poderia enquadras, como no desenvolvimento de aplicações móveis para a saúde.

Sendo assim, argumenta-se que há a necessidade de «agilizar» o processo de desenvolvimento de *software* para a indústria médica, à luz dos novos regulamentos. Pretende-se um modelo de ciclo de vida compatível com as normas e *standards* necessários para adquirir as certificações que qualificam o produto como dispositivo médico, mas também que permita flexibilidade na reação à mudança proporcionada pelo *agile*. No próximo capítulo, apresentam-se soluções da bibliografia com trabalhos neste sentido.

Tabela 3.4: Condições favoráveis para metodologias ágeis e *plan-driven*

Condição	***Agile***	***Plan-driven***
Ambiente de mercado	Mudanças frequentes das preferências do cliente;	Condições de mercado estáveis e previsíveis;
Envolvimento do cliente	Colaboração próxima e rápido *feedback* é possível;	Cliente indisponível para esclarecimentos frequentes. Requisitos são claros e há pouca probabilidade de se alterarem;
Tipo de Inovação	Problemas técnicos complexos com soluções desconhecidas, sem âmbito definido.;	Existe experiência no tipo de solução pretendida. Especificações detalhadas e planeamento podem ser feitos com confiança.
Modularidade do trabalho	Desenvolvimento incremental tem valor, na medida em que a solução pode ser usada pelo cliente entre cada entrega;	O cliente não pode iniciar testes de aceitação até que o produto esteja concluído;
Impacto dos erros	Podem potenciar oportunidades de aprendizagem;	Podem ter consequências catastróficas;

4

Estado da Arte do Processo de Desenvolvimento de *Software* para Dispositivos Médicos

O processo de engenharia de *software* na indústria médica aplicada ao desenvolvimento de dispositivos médicos é o principal foco deste documento. Neste capítulo apresenta-se uma revisão da literatura, que pretende ser representativo do estado atual do mesmo. Este tema encontra-se em forte desenvolvimento, e a pesquisa efetuada permite concluir com segurança que não existe um *standard* definido para a aplicação do modelo ou do processo mais adequado. Aliás, em 2007, Christian Denger *et al* [72] documentaram que cerca de metade das empresas representadas no seu estudo nem sequer seguiam modelos do ciclo de vida de desenvolvimento definidos, o que é indicativo da pertinência deste estudo.

Neste capítulo, começar-se-á por analisar a bibliografia existente no que diz respeito à aplicação de modelos específicos para o ciclo de vida de desenvolvimento, como é o caso do *waterfall* e das metodologias *agile* como o *scrum* e o XP, aplicados ao desenvolvimento de *software* aplicado dispositivos médicos. Haverá uma incidência particular e propositada para as metodologias *agile*, uma vez que têm um caráter mais inovador no setor dos sistemas críticos. Seguir-se-á um estudo mais particular por atividades e tópicos especialmente relevantes que foram documentadas em isolamento. É o caso da atividade de gestão do risco e *quality assurance*.

4.1 Aplicação de Modelos do Ciclo de Vida

4.1.1 Abordagens *Plan-Driven*

A indústria dos sistemas críticos tem um historial comprovado com as metodologias orientadas à planificação, tal como detalhado no capítulo 3.2.1. Mais particularmente na indústria dos dispositivos médicos, Suzanne e Robert Leif [73] foram dos primeiros a abordar a questão do ciclo de vida de desenvolvimento de *software* no sentido de ir ao encontro aos regulamentos e *standards* da indústria. Os autores apresentaram um estudo compreensivo sobre um processo orientado a produção de documentação, enumerando os principais artefactos necessários que deverão ser incluídos no ciclo de vida de desenvolvimento para satisfazer as autoridades reguladoras, sugerindo que os modelos em espiral [55] ou *waterfall* [48] poderiam ser adotados.

O artigo supramencionado já data de há mais de duas décadas, porém, os modelos lineares tradicionais continuam a ter alguma tração na comunidade científica. Zema *et. al* [11] apresentou, em 2015, uma *framework* para o desenvolvimento de *software* baseada no modelo *waterfall*, a fim de promover a produção de *software* confiável e com segurança especificamente para dispositivos médicos. A estratégia dos autores consistiu em mapear cada regulamento/*standard* relevante a uma atividade do ciclo de desenvolvimento, sugerindo, para cada uma, ferramentas UML úteis para a produção de documentação em cada atividade. É feita particular incidência na diretiva 93/42/EC [21], ainda em vigor à data e as atividades incluídas no processo são as convencionais para este tipo de modelo, nomeadamente levantamento de requisitos, análise de requisitos, desenho do sistema, implementação, validação, instalação e manutenção seguidas de uma forma sequencial e às quais se acrescentaram atividades mais específicas para o âmbito dos dispositivos médicos, tais como instalação de um sistema de gestão de qualidade, definição de classe de risco e avaliação por um órgão notificado.

4.1.2 Abordagens *Agile*

De uma forma geral, as metodologias *agile* são entendidas como não sendo as mais apropriadas para o desenvolvimento em ambientes fortemente regulados. Aliás, na bibliografia encontra-se alguns exemplos de publicações sobre os principais desafios das metodologias ágeis neste tipo de projetos. O grupo de investigação sobre *software* regulado no Instituto de Tecnologia de Dundalk na Irlanda, tem-se debruçado sobre esta matéria no âmbito do desenvolvimento de dispositivos médicos em anos recentes. A principal barreira na adoção de abordagens ágeis é, de acordo com estes autores, a necessidade de cumprir com as auditorias e controlos por parte das entidades reguladoras, nomeadamente no que diz respeito à produção da documentação necessária [52, 70, 74]. Outros desafios encontrados incluem dificuldades em implementar estratégias de rastreamento entre os artefactos, falta de planeamento e dificuldade em enquadrar as entregas incrementais características dos modelos ágeis num ambiente *safety-critical*, isto porque é impraticável entregar *software* não funcional se as vidas dos utilizadores dependerem dele [70].

Ainda assim, no trabalho dos mesmos autores são reforçadas também as insuficiências dos modelos *plan-driven* no que toca a responder às necessidades dos projetos de *software* para o fim em discussão, nomeadamente a incapacidade de acomodar mudanças nos requisitos [74]. Além disso, estes investigadores consideram que todas as dificuldades enumeradas relativamente às metodologias *agile* são ultrapassáveis, por exemplo, a incapacidade destes métodos para produzir a documentação necessária é uma interpretação errada das filosofias ágeis, sendo que não há nenhuma barreira concreta para que isto não seja possível [70].

As diferentes formas de combater estas dificuldades é precisamente um dos principais temas de interesse deste documento. Com as alterações ao panorama legal europeu discutidas na secção 2.3, as metodologias ágeis tornam-se mais relevantes e vantajosas para o desenvolvimento de *software* aplicado a dispositivos médicos e, para tornar isto possível, é necessário perceber como é que estes modelos se podem adaptar a cenários mais regulados e críticos.

Já em 2002, quase imediatamente após o surgimento do manifesto *agile* [56], Bowers *et. al* [75] estudaram formas de fazer *tailoring* [*] ao modelo de *Extreme Programming* [59] para o adaptar ao desenvolvimento de sistemas críticos, neste caso, um sistema de comunicação para a segurança pública. As adaptações incluíram estender as interações de desenvolvimento para garantir que as entregas incluíam todas as *features* necessárias para garantir funcionalidade, fazer o desenho das funcionalidades para ter em conta o conjunto total de requisitos (em vez de desenhar apenas tendo em conta os requisitos para a próxima *milestone*, como é recomendado pelo XP) e limitar a liberdade para práticas de *refactoring*. Ainda assim, o grupo foi bem sucedido a implementar práticas ágeis importantes, nomeadamente integração

[*] *Tailoring*, cuja melhor tradução será talvez «costumização», é um termo usado em engenharia de software que significa um processo no qual são feitas adaptações de forma a determinar uma abordagem específica mediante a situação do projeto.

contínua, programação a pares e automatização total de testes para aumentar a sua produtividade global.

O trabalho supramencionado quebrou o gelo para que a comunidade científica aprofundasse a investigação relativamente à aplicação do *agile* em projetos de natureza *safety-critical*, inclusivamente com *software* embebido [76]. Especificamente na indústria dos dispositivos médicos, Jon Spence [77] publicou, em 2005, um trabalho que viria a virar uma página, uma vez que expressou o descontentamento pelo processo rigoroso imposto pelo modelo *waterfall*. Spence era engenheiro sénior na Medtronic, uma empresa importante no desenvolvimento de *software* aplicado a dispositivos médicos de Classe III e escreveu sobre a adoção do *agile* para alterar a cultura de atenção exagerada a especificação e a estimativas antecipadas. O autor reporta que a mudança se mostrou frutífera em vários aspetos, mas principalmente no entusiasmo e contentamento demonstrado pelas equipa de desenvolvimento em trabalhar com práticas como *daily scrum*, integração contínua e *simple design*.

Nos anos seguintes, foi notório o aumento da adoção dos métodos ágeis no âmbito empresarial. Em 2008, engenheiros da Cochlear Limited [78] (conhecida pelos seus dispositivos implantáveis para tratamento de problemas de audição) descreveram a sua experiência com *scrum* num ambiente de desenvolvimento sujeito a *standards* como a IEC 62304 [17] e a ISO 14871 [32]. Os autores identificaram como principais desafios a obrigatoriedade de ter uma estratégia de rastreabilidade entre os artefactos e a disparidade entre os processos *waterfall* utilizados pelos restantes departamentos da empresa (nomeadamente para desenvolvimento de hardware) e o departamento de *software*. Esta publicação espelha de uma forma bastante exemplificativa as ineptidões dos métodos ágeis ao lidar com *software* embebido. Os investigadores descrevem que, a necessidade de integrar *software* com *hardware* obrigava a um volume significativo de testes manuais, que por sua vez se tornavam um *bottleneck* para a conclusão das *user stories* (cada *story* apenas pode ser classificada como concluída após testada com sucesso). Ainda assim, os autores classificam a experiência como produtiva.

No ano seguinte, Rasmussen *et. al* [79] da Abbott, empresa multinacional ligada à tecnologia da saúde, apresentaram mais um estudo sobre a adoção de um modelo ágil num ambiente fortemente regulado, neste caso, para dispositivos médicos de Classe III. Este trabalho é bastante demonstrativo do potencial do *agile* para otimizar o desenvolvimento de dispositivos médicos quando se trata de *software stand-alone*, dado que fala de uma solução que auxilia no diagnóstico e monitorização de progressão de doenças associadas ao genoma dos pacientes. Os autores adotaram um modelo chamado Agile+, desenvolvido pela empresa AgileTek. O modelo incluiu iterações curtas de 6 a 8 semanas, testes automáticos, integração contínua, reuniões diárias de equipa e retrospetivas realizadas por *stakeholders*, práticas que não eram usadas no processo anterior desta equipa da Abbott. Concluiu-se que a mudança no processo de desenvolvimento levou a uma redução do custo do projeto, assim como na sua duração; porém, convém notar que o artigo não inclui considerações sobre estratégias de gestão de risco e de documentação, ambas obrigatórias para obedecer aos padrões das entidades reguladoras.

Ge *et. al* [80] fizeram, em 2010, uma publicação que tenta identificar, aplicar e avaliar práticas existentes em engenharia de segurança e fazer adaptações às mesmas de forma a que possam «agilizar» os sistemas críticos. Os autores apresentaram um modelo iterativo de desenvolvimento para *software safety-critical* que pretende focar em dois aspetos importantes, encarados como impedimentos da implementação de modelos ágeis neste âmbito:

1. Nos modelos ágeis, a fase de planeamento em cada iteração é normalmente um desenho incompleto, que se foca bastante em abstrações e em arquitetura abstrata, em detrimento de uma planificação detalhada. Contudo, para projetos de natureza crítica o planeamento necessita de ter detalhe suficiente para que possa ser feita uma análise de risco. Os autores sugerem aumentar o nível de rigor no planeamento em cada iteração, de forma a que possa ser produzida uma especificação de requisitos funcionais para cada componente incluído no incremento seguinte, assim como um modelo de arquitetura. Sugerem também que a análise de risco seja incluída nesta fase de planeamento, o que é uma ideia bastante interessante, uma vez que se criam as condições para que esta análise seja atualizada e estendida em cada iteração;

2. Uma das principais características do *agile* são as entregas iterativas. Isto constitui um problema para o desenvolvimento de *software* crítico uma vez que é difícil garantir a segurança dos sistemas sem que todos os requisitos estejam implementados e também devido às necessidades de certificação. Os autores sugerem uma abordagem baseada em argumentação de segurança. O que classifica uma entrega como aceitável é baseado no nível de integridade do *software*, no *standard* que permite a certificação desejada, nos requisitos da autoridade reguladora e na análise de risco. O objetivo será construir iterativamente o produto de *software*, mas também o argumento de segurança.

Nos anos seguintes, o grupo de investigação já mencionado do Instituto de Tecnologia de Dundalk, liderado por Martin McHugh, apresentaram uma série de artigos nos quais descreveram a integração de práticas ágeis no ciclo de desenvolvimento para dispositivos médicos. o objetivo da investigação dos mesmos foi a criação de um modelo para o ciclo de desenvolvimento de *software* capaz de agregar as principais vantagens das metodologias ágeis, mas também capaz de produzir os documentos necessários para satisfazer as entidades reguladoras e conseguir as certificações necessárias.

Em [81, 82], os autores descrevem um modelo híbrido (que tem características ágeis, mas também do modelo em cascata) baseado no *V-Model*, que foi sujeito a *tailoring* de forma a maximizar o seu impacto especificamente no ecossistema dos dispositivos médicos. Este modelo integrou as conclusões apresentadas por Ge *et al.* [80] ao modelo em V. Além disso, foram adicionadas práticas ágeis como *sprints*, *daily sand-ups*, retrospetivas e *user stories*. Este trabalho foi estendido no ano seguinte [83], onde os autores aprofundaram o estudo sobre as dificuldades das empresas que desenvolvem para a área da saúde e salientaram que a maioria das empresas se queixava da dificuldade de acomodar a mudança de requisitos

em fases avançadas do processo. Além disso, foi apresentado novamente o modelo em V adaptado para acomodar 13 práticas ágeis distintas.

Esta adaptação do *V-Model*, apelidada de *AV-Model*, foi apresentada formalmente em [14], de onde foi retirada a figura 4.1.

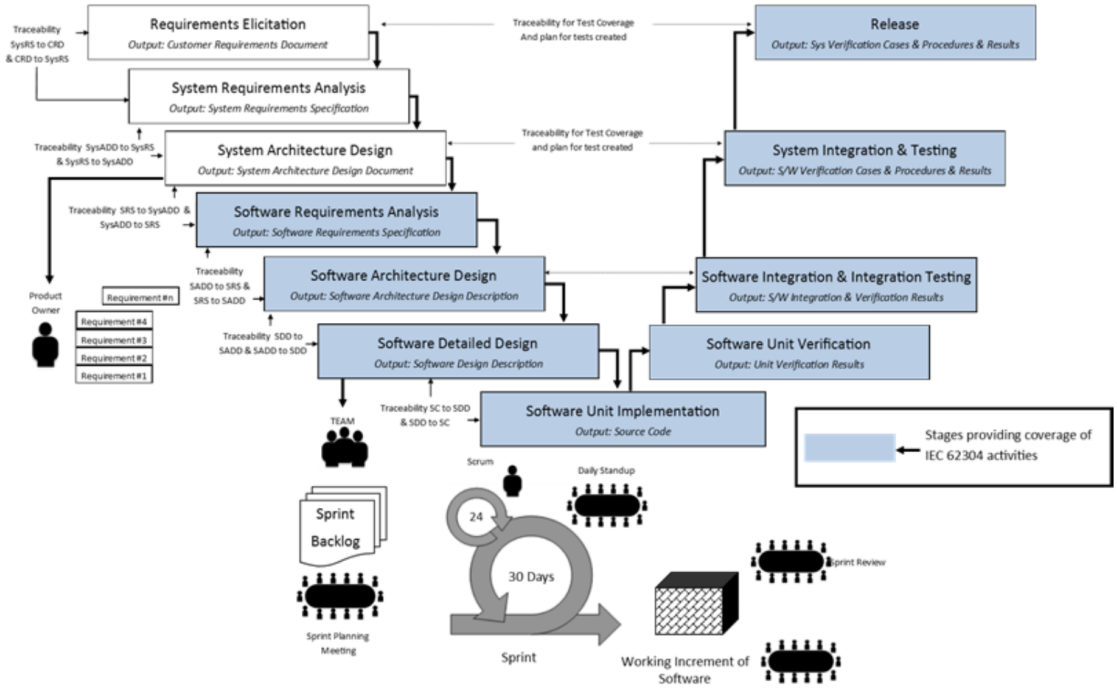


Figura 4.1: O *AV-Model*. Retirado de [14]

O modelo apresentado tem como componente base uma estratégia iterativa, de forma a facilitar alterações consequentes de mudanças nos requisitos. As atividades de levantamento de requisitos (que pode, também, ter iterações) e arquitetura/desenho do sistema são realizadas apenas uma vez por projeto, como sugerido pelo modelo *waterfall*. Por outro lado, atividades de análise de requisitos, desenho detalhado e testes são realizadas para cada requisito de forma iterativa e incremental e, além disso, implementação e testes unitários são realizados por cada item de *software*. Os autores concluíram que a integração da filosofia *agile* na cultura de desenvolvimento numa organização que produzia *software* para dispositivos médicos resultou num decréscimo de custos associados à acomodação de mudanças de âmbito inesperadas.

Em 2013, Fitzgerald *et al.* [84] endereçaram a implementação de um modelo ágil na empresa QUMAS, uma empresa irlandesa que desenvolve modelos de *compliance* para diversas organizações. Como tal, esta empresa necessita de obedecer a várias normas internacionais, algumas delas comuns ao desenvolvimento de dispositivos médicos como a ISO/IEC 12297 [18] e a ISO 13485 [27]. Os autores apelidaram

a este modelo *R-Scrum*, um diminutivo de *regulated scrum* e este consiste numa adaptação do *scrum* para satisfazer os requisitos dos ambientes regulamentados (figura 4.2). Uma adaptação significativa neste modelo é no processo de *quality assurance*: existe uma equipa independente que faz auditorias no final de cada *sprint* e, caso haja inconformidades, estas são transpostas para o *backlog* do produto.

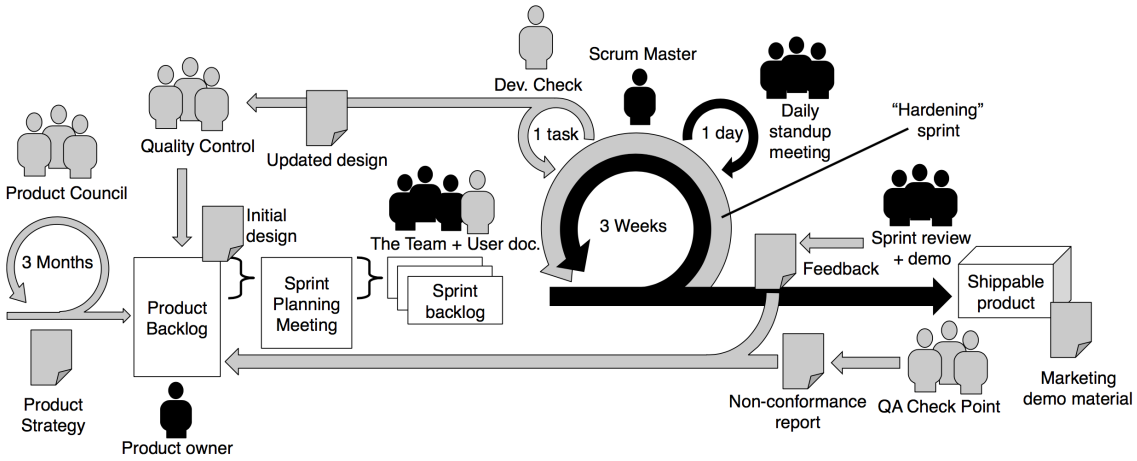


Figura 4.2: O *R-Scrum*. Retirado de [84].

Em 2016, Pikkarainen *et al.* [25] apresentaram uma *framework* para a avaliação do processo de desenvolvimento capaz de integrar práticas ágeis chamada *MDevSPICE*. Esta ferramenta foi desenhada para permitir o desenvolvimento de *software* para dispositivos médicos de forma segura e *compliant* com os regulamentos em vigor. Em primeiro lugar, esta *framework* fornece um modelo de referência para o *software*, ou seja, um conjunto de processos estruturados e normalizados que integram requisitos de vários *standards*, especificamente aqueles aplicáveis aos dispositivos médicos; isto inclui ciclo de vida do sistema, do *software* e atividades paralelas como gestão de configurações. Em segundo lugar, a ferramenta fornece um modelo de avaliação baseado nos artefactos produzidos por cada processo ou atividade. Os autores propuseram a integração de metodologias ágeis nesta *framework*, sugerindo que poderia estimular a produção de melhores resultados globais. Esta *framework* ainda se encontra em investigação e expansão [85].

4.2 Estudos sobre Atividades Específicas do Ciclo de Vida

Se há uma conclusão que se pode tirar da secção anterior, é que dificilmente se conseguiria aplicar um modelo *agile* como *scrum* a um ambiente fortemente regulamentado como o do desenvolvimento de dispositivos médicos de uma forma *out of the box*, ou seja, sem fazer qualquer ajuste [86]. O *tailoring* é, assim, um conceito essencial neste âmbito, e que pode ser aplicado de diferentes maneiras. Nesta secção,

o foco será de analisar trabalhos da bibliografia que se debrucem sobre a importância e o *tailoring* de atividades que podem ser consideradas críticas numa transição de uma metodologia *plan-driven* para *agile* e que são obrigatórias para a *compliance* mediante normas internacionais.

4.2.1 *Quality Assurance*

Quality assurance, ou simplesmente QA, não é propriamente uma atividade, mas sim o conjunto de processos que visa assegurar que o produto desenvolvido vai de encontro com as especificações que foram definidas. Ainda assim, é um conceito de relevo para este documento uma vez que é de uma importância decisiva quando falamos de sistemas críticos. Esta crença não é recente na comunidade do *software*, e é acentuada pelo influente trabalho de Dolores Wallace e Richard Kuhn [4], que analisaram um conjunto de falhas relacionadas com *software* especificamente aplicadas a dispositivos médicos. Os autores concluíram que a utilização metódica de práticas de qualidade é significativa para a redução de falhas associadas ao *software*. Também é referido que não existe nenhuma solução única e transversal para as práticas de *quality assurance*, sendo que deve ser desenvolvido um conjunto de estratégias que vão de encontro com as características da organização e do projeto.

Mais recentemente, em 2016, Bujok *et al.* [87] apresentaram uma abordagem bastante completa para a integração dos *standards* internacionais assim como regulamentos e diretivas associadas aos dispositivos médicos num sistema de gestão de qualidade. A abordagem apresentada pelos autores encontra-se ilustrada na figura 4.3.

4.2.2 Gestão do Risco

A gestão do risco inclui a identificação, avaliação e prioritização dos riscos associados ao funcionamento do *software*, seguido da coordenação dos recursos existentes para minimizar e monitorizar o impacto dos mesmos. Por sua vez, o risco é definido pela norma ISO 13485 [27] como o efeito da incerteza nos objetivos.

Esta atividade é especialmente fundamental na área da saúde e dos dispositivos médicos e os fabricantes dos mesmos devem endereçar o risco para com os pacientes, operadores e profissionais de saúde e também para com o ambiente. Estes aspetos foram realçados por Steven Rakitin [88], que endereçou a o espectro do risco associado ao desenvolvimento para dispositivos médicos, assim como as melhores práticas para o desenvolvimento de um planeamento de gestão de risco (da forma exigida pela norma IEC 62304 [17]). O autor realça a importância de ter um modelo de ciclo de vida robusto para o desenvolvimento de software de forma a minimizar o risco, e isso inclui a inclusão de atividades de gestão de risco no decorrer de todo o ciclo de vida do produto, desde o início do projeto até à fase de manutenção.

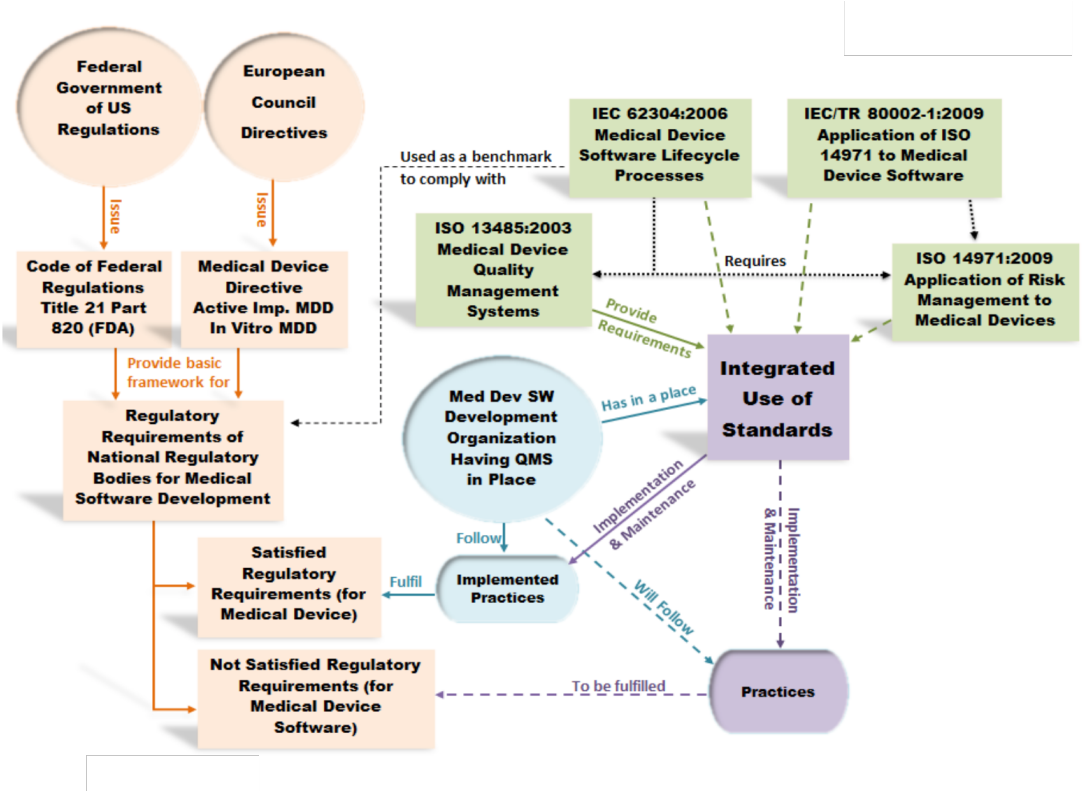


Figura 4.3: Abordagem para a integração dos requisitos impostos na ISO 13485 [27], IEC 62304 [17] e ISO 14971 [32] e a sua implementação dentro processo de desenvolvimento e sistema de gestão da qualidade. Retirado de [87].

4.2.3 Rastreabilidade

O aumento da complexidade do *software* associado a dispositivos médicos gerou a necessidade de uma rastreabilidade eficiente associada ao processo de desenvolvimento. Além disso, a rastreabilidade é um requisito imposto pelas normas associadas ao processo de ciclo de vida para dispositivos médicos, nomeadamente pela IEC 62304 [17]. A rastreabilidade é a propriedade de um artefacto ou elemento de código que indica se este pode ser relacionado com a sua origem; por exemplo, um teste deve ter rastreamento para os requisitos que pretendem validar e que são o motivo da sua existência.

O *Med-Trace* [89, 90, 91, 92] é um método de avaliação e melhoramento do rastreamento no processo de desenvolvimento de *software* aplicado a dispositivos médicos. O método é baseado nas melhores práticas de engenharia e nas recomendações internacionais como a ISO/IEC 15504 [30], assim como nas regulamentações e normas aplicáveis a esta área. O *Med-Trace* já foi implementado na indústria com sucesso e demonstra a importância que esta atividade pode representar na área dos dispositivos médicos, devido à sua criticalidade.

Parte II

Solução Proposta

5

Uma Abordagem Ágil para o Desenvolvimento de *Software* para Dispositivos Médicos

---

Como foi detalhado no capítulo 2, a introdução do Regulamento para Dispositivos Médicos significou uma alteração estrutural para a definição de dispositivos médicos com consequências entusiasmantes para o desenvolvimento de *software* certificado. Tornou-se possível desenvolver soluções com características drasticamente diferentes daquilo que era possível até então e este facto justifica alterações ao processo de desenvolvimento, nomeadamente ao seu ciclo de vida.

Produtos mais *lightweight*, como aplicações móveis, justificam a introdução de metodologias que estimulem flexibilidade, rápida evolução e reação a mudanças tais como as metodologias ágeis. No estudo bibliográfico apresentado no capítulo 4, é notório que, apesar da indústria do sistemas críticos ainda não ter adotado ciclos de vida *agile* de uma forma generalizada, têm sido feitas tentativas bem sucedidas de introduzir abordagens ágeis no processo de desenvolvimento.

Pretende-se, neste capítulo, introduzir um modelo de ciclo de vida de desenvolvimento ágil desenhado para responder às exigências feitas pela indústria dos dispositivos médicos, espelhadas na norma IEC 62304:2006, sem desvirtuar as filosofias *agile*. Quando se fala em ciclo de vida, especialmente em modelos baseados no *agile*, não se fala exclusivamente na sequência de processos subjacentes ao mesmo, mas também em todas as estratégias, atividades e recursos humanos que permitem atingir os objetivos propostos.

Sendo assim, neste capítulo discutem-se os **processos** que compõem o ciclo de vida proposto, as **cerimónias** que inclui e onde se inserem, os **papéis e responsabilidades** de cada elemento da equipa e, finalmente, quais são as **estratégias de desenvolvimento** e gestão indispensáveis para implementar o modelo com sucesso.

5.1 Descrição do Ciclo de Vida

Em engenharia de software, *tailoring* é uma atividade que permite ajustar o processo ou ciclo de vida de desenvolvimento de acordo com as necessidades do projeto ou do contexto [93]. Este conceito é basilar para o sucesso de qualquer modelo de desenvolvimento de *software*, uma vez que cada negócio tem as suas particularidades e as suas necessidades específicas. O que se sugere neste capítulo é um modelo de ciclo de vida adaptado da metodologia *SCRUM* introduzida por Ken Schwaber and Jeff Sutherland [60].

Uma ilustração do modelo sugerido encontra-se apresentada na figura 5.1.

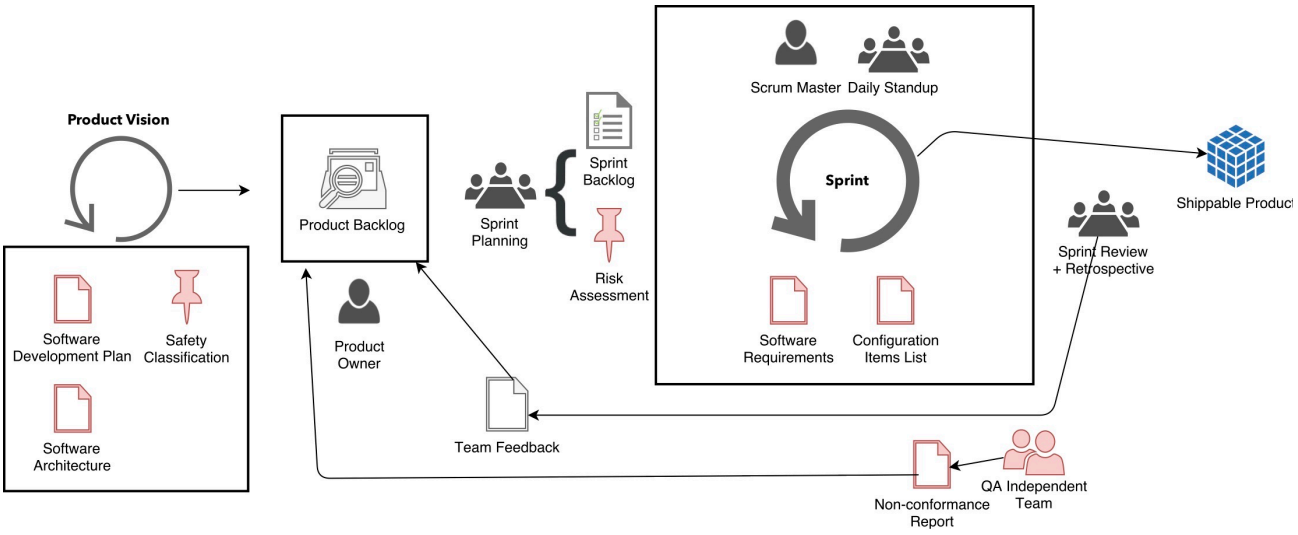


Figura 5.1: Modelo de Ciclo de Vida proposto, adaptado do *SCRUM*. O modelo tem como objetivo principal fornecer uma estrutura para *compliance* de acordo com a norma IEC 62304:2006 e conciliar isso com as características de flexibilidade e reação à mudança das metodologias ágeis. Todos os itens coloridos a vermelho não estão contemplados no ciclo de vida *SCRUM* clássico, sendo que são objetos de *tailoring*.

5.1.1 *Product Vision*

Esta atividade, sem duração pré-definida, pode ser traduzida literalmente como «visão do produto» e ocorre normalmente durante a fase de proposta do projeto, no caso de existir um cliente definido.

No decorrer da *product vision*, o *Product Owner*, o *SCRUM Master* e (caso se justifique) elementos da equipa de desenvolvimento com a responsabilidade de fazer o desenho de interfaces gráficas (*UX Designers)* trabalham em conjunto de forma a criar uma ideia global para o produto. Deve ser conduzida uma análise macro de requisitos de forma a ganhar compreensão da solução, desenvolvidos os conceitos principais de arquitetura do sistema (desenho do sistema) e também desenhados *mockups* [*] das principais interfaces gráficas, caso aplicável.

Esperam-se os seguintes resultados desta atividade:

1. A **classificação de segurança do dispositivo médico**. O dispositivo médico deve ser classificado como A, B ou C mediante as consequências da sua falha.

2. Um **documento de planificação de software** (DPS). Este documento deve mostrar evidências das planificações feitas relativamente ao processo de desenvolvimento. Deve mencionar o ciclo de

[*] *Mockups* representam demonstrações gráficas do produto. Refletem as escolhes de desenho, esquemas de cores, tipografia e visual de navegação.

vida usado, quais são os *deliverables* [†] previstos, quais são os *standards* e ferramentas a utilizar no decorrer do desenvolvimento, assim como detalhar as planificações relativamente a estratégias de teste, verificação, gestão de risco e gestão de configurações.

3. Um ***backlog*** [‡] **do produto**, que espelhe um conjunto de requisitos de *software*.

4. Um **documento de arquitetura de *software*** (DAS). Com base no conhecimento adquirido para construir o *backlog* do produto, este documento deverá detalhar todos os componentes e interfaces do sistema. Não é necessário que este *deliverable* seja um documento escrito, poderá ser construído através de ferramentas como *Enterprise Architect* [94].

5.1.2 Sprint 0

Ao contrário das *sprints* tradicionais, esta sprint é facultativa e não inclui as cerimónias do *SCRUM*. É dedicada inteiramente à revisão e verificação dos *outcomes* da *product vision*. Especificações de interfaces e provas de conceito devem ser realizadas de forma a validar a arquitetura proposta para o sistema e, nesta medida, a equipa de desenvolvimento já deve estar toda disponível nesta atividade.

Como resultado desta fase, deve existir uma revisão do *backlog* do produto, documentação atualizada de planeamento e arquitetura e também a infraestrutura de desenvolvimento (servidor de integração contínua, configurações para o ambiente integrado de desenvolvimento, infraestrutura de teste). Outra consequência importante do *sprint 0* é a *definition of Done*, que deverá ser compreendida por toda a equipa do projeto e igualmente aplicada em todos os *sprints* subsequentes.

5.1.3 Sprint

Cada *sprint* é uma iteração de desenvolvimento, com uma duração variável entre duas e quatro semanas. Esta duração pode variar de acordo com as características únicas do dispositivo médico ou do projeto em si, contudo, deve ser estabelecida na etapa de *product vision* e registada no DPS. Cada *sprint* tem um objetivo (*sprint goal*), daquilo que deve ser construido e no final da mesma deve existir uma solução funcional.

Em cada *sprint*, a equipa de desenvolvimento tem o objetivo de trabalhar para um incremento do produto, significando que é necessário desenvolver a funcionalidade prevista e realizar todas as atividades de teste e integração de forma a obter uma versão funcional da solução de *software* no final do *sprint*.

[†]Um *deliverable* ou «entregável» é um bem ou serviço produzido como consequência de um projeto relativamente ao qual há a expectativa de que seja entregue a um cliente. No contexto desta dissertação, este termo é utilizado vulgarmente para a documentação de entrega obrigatória.

[‡]Um *backlog* é uma listagem de funcionalidades ou tarefas técnicas mantidas pela equipa e que, a cada momento, são necessárias para concluir um projeto ou uma entrega do produto.

Quando uma *sprint* se inicia, elementos externos à equipa de desenvolvimento não devem interferir no mesmo, sendo que é o papel do *SCRUM Master* proteger a equipa de eventuais interferências. Todos os pedidos de alteração (*change requests*) deverão ser geridos pelo *SCRUM Master* e apenas aplicáveis em *sprints* futuras, na medida em que devem ser introduzidos no *backlog* do produto.

Uma *sprint* encontra-se finalizada («*Done*») quando todos os itens estão finalizados de acordo com a *definition of Done*, quando todos os critérios de aceitação estão cumpridos e o quando objetivo da *sprint* está alcançado.

5.1.4 *QA Checkpoint*

No final de cada *sprint*, uma equipa de *quality assurance* independente da equipa de desenvolvimento deve realizar uma auditoria interna com o objetivo de fazer uma verificação dos principais *deliverables* e evidências de forma a garantir *compliance* com a norma IEC 62304:2006. Como resultado deste processo de auditoria, deve ser elaborado um relatório de não conformidades que deve ser entregue ao *Product Owner*, para que sejam criadas medidas de correção. Estas medidas devem ser transformadas em tarefas e incluídas no *product backlog*, para que as não conformidades sejam corrigidas.

Esta estratégia é baseada no modelo *R-SCRUM* [84].

As auditorias internas apresentam inúmeras vantagens como mecanismos de *quality assurance*, porque ajudam a explorar problemas tecnológicos e de negócio. A equipa de QA, neste caso específico, deverá manter-se atualizada com as exigências legais no setor dos dispositivos médicos, de forma a conseguir monitorizar riscos e encontrar anomalias. Esta equipa deve questionar o *status quo*, identificar áreas problemáticas no processo e contribuir para poupar tempo e recursos.

Este mecanismo de auditoria interna deverá, também, introduzir confiança de que um eventual processo de avaliação por uma entidade externa será bem sucedido. Para orientação da construção do relatório de não conformidades supramencionado, a equipa de QA deve utilizar uma *checklist* de auditoria criada pela organização para o efeito.

No Anexo B encontra-se um exemplo deste documento, orientado para o cumprimento das indicações contidas na norma IEC 62304:2006. Este documento foi construído através de uma análise detalhada à norma em questão e deve ser utilizado da seguinte forma:

1. Cada secção da *checklist* corresponde (não linearmente, mas de uma forma lógica) a um capítulo da norma IEC 62304:2006. O auditor deve, para cada ponto de cada secção, procurar evidências irrefutáveis do cumprimento. Apenas devem ser considerados os pontos correspondentes à classe de segurança do dispositivo. Cada ponto tem a indicação das classes às quais se aplica;

2. Em consequência das evidências encontradas, deve assinalar se o ponto permite cumprimento ou não cumprimento, assim como indicar eventuais sugestões de melhoria e observações;

3. O documento permite fazer uma análise estatística percentual do nível de cumprimento da *sprint* que deve ser analisado de forma a encontrar tendências e melhorar o processo;

4. Com base nesta análise, deve ser produzido um relatório de não-conformidades, que por sua vez deve ser entregue ao *Poduct Owner* para que as mesmas sejam prioritizadas e introduzidas sob a forma de itens de configuração na próxima *sprint*.

Adicionalmente, este documento promove a obtenção dos seguintes objetivos:

1. Orientar os elementos da QA team no processo de auditoria interna, fornecendo uma plataforma de fácil utilização;

2. Dividir e identificar claramente quais são aos pontos da norma para cada classe de segurança do dispositivo médico. A equipa de QA apenas deve procurar evidências para os pontos que correspondem às classes de segurança do dispositivo em questão no projeto auditado;

3. Fazer o mapeamento entre cada ponto da *checklist* e o ponto da norma IEC 62304:2006 para fácil consulta e justificação de eventuais não-conformidades;

4. Facilitar a construção de um relatório de não-conformidades (a entregar ao *Product Owner*) através de um ambiente onde o auditor pode assinalar não só estas não-conformidades, mas também observações, sugestões de correção e melhoria.

5.2 Cerimónias SCRUM

As reuniões, eventos ou cerimónias são uma componente estrutural do desenvolvimento ágil. Ajudam a capacitar a equipa e a implementar corretamente o ciclo de vida *SCRUM*.

5.2.1 *Sprint Planning*

Cada *sprint* inicia com uma reunião de planeamento. Esta cerimónia tem como objetivo a definição do âmbito da *sprint* e a descriminação das tarefas que permitem alcançar o mesmo. O *Product Owner* tem a responsabilidade de apresentar um *backlog* do produto ordenado por prioridade e atualizado de forma a que a equipa possa fazer estimativas relativamente a cada tarefa a incluir na *sprint* (*backlog* da *sprint*).

Adicionalmente, nesta cerimónia, o *Product Owner* e o *SCRUM Master* devem identificar os riscos que dizem respeito aos itens de maior prioridade no *backlog* e apresentá-los à equipa de desenvolvimento

para que possam ser avaliados novamente. Pretende-se que o risco seja gerido de uma forma iterativa, da mesma forma que o produto. No início de cada *sprint*, a equipa deve reanalisar e atribuir novas prioridades aos riscos que afetam o projeto, dando especial ênfase aos itens que serão introduzidos na nova *sprint*. Desta forma, constrói-se uma argumentação de segurança cada vez mais sólida [80].

5.2.2 *Sprint Review*

A reunião de revisão da *sprint* serve o propósito de fornecer um ambiente colaborativo para analisar o valor que foi construído no decorrer da *sprint* e debater sobre quais são as próximas tarefas a alcançar. Esta cerimónia fornece informação importante ao *Product Owner*, para que este possa atualizar o *backlog* do produto e preparar a próxima reunião de *sprint planning*.

5.2.3 *Sprint Retrospective*

O objetivo principal da reunião de retrospetiva é que a equipa e o *SCRUM Master* reflitam nos acontecimentos da última *sprint*, para que possam melhorar o processo de desenvolvimento na seguinte. A intenção deverá ser tornar sempre as *sprints* cada vez mais eficientes, mas também mais satisfatórias para todos os envolvidos.

Esta cerimónia deverá ser realizada depois da *sprint review* mas antes do *sprint planning* e deve culminar com um conjunto de medidas concretas que possam ser introduzidas no *backlog*, a fim de melhorar a experiência de desenvolvimento na *sprint* seguinte.

5.2.4 *Daily SCRUM*

Consiste numa reunião que deve acontecer diariamente e à mesma hora, para que todos os elementos da equipa sejam atualizados relativamente ao progresso das tarefas de cada elemento. Tem o objetivo de coordenar toda a equipa e estimular a comunicação entre todos. Deve ter uma duração máxima de quinze minutos, no decorrer dos quais todos os elementos devem estar de pé (*stand-up meeting*).

5.3 Papeis e Responsabilidades

5.3.1 *SCRUM Team*

A equipa *SCRUM* é constituída pela equipa de desenvolvimento, pelo *SCRUM Master* e pelo *Product Owner*. A equipa é auto-organizada, na medida em que é responsável por escolher a melhor forma de concretizar os seus objetivos, independentemente de outros elementos externos à mesma.

5.3.2 *SCRUM Master*

O *SCRUM Master* tem a responsabilidade de garantir que o ciclo de vida *SCRUM* é entendido e concretizado por toda a equipa.

As responsabilidades que este elemento tem relativamente à equipa de desenvolvimento inclui ensinar e treinar a mesma para que se possa auto-organizar, remover impedimentos ao progresso da equipa e facilitar e organizar as cerimónias *SCRUM*.

5.3.3 *Product Owner*

O *Product Owner* tem a responsabilidade de maximizar o valor do produto e a eficiência da equipa de desenvolvimento. É também o único indivíduo responsável pela gestão do *backlog* do produto. Para tal, deve:

- Expressar claramente os itens no *backlog* do produto;
- Ordenar estes itens por prioridade;
- Garantir que os itens são visíveis, transparentes e claros para toda a equipa de desenvolvimento;

Outra das grandes responsabilidades do *Product Owner* deverá ser a de monitorização do *feedback* relativamente ao produto, assim como gestão dos pedidos de modificação (*change requests*). Esta entidade deverá analisar e aprovar qualquer pedido de modificação que seja feito por parte dos *stakeholders* e modificar o *backlog* do produto em concordância. Além disso, deve comunicar e gerir as expectativas dos mesmos.

É importante que as responsabilidades do *Product Owner* não coincidam com as do *SCRUM Master*, uma vez que estes papéis são incompatíveis por definição. O *Product Owner* pode ser o cliente, dependendo da natureza do projeto.

5.3.4 Equipa de Desenvolvimento

A equipa de desenvolvimento é constituída por profissionais que trabalham com o objetivo de entregar um incremento funcional (de acordo com a definição de *Done*) do produto no final de cada *sprint*. Os elementos da equipa devem ser pluridisciplinares, na medida em que o grau de especialização dos seus elementos deve ser reduzido.

A equipa deve ser auto-organizada, na medida em que ninguém lhes deve dizer como converter o *backlog* da *sprint* em *software*. Em concordância, o objetivo da *sprint* é da responsabilidade da equipa como um todo.

5.3.5 *QA Team*

A equipa de *quality assurance* não faz parte da *SCRUM Team* e não tem qualquer obrigação de ser específica por cada projeto. Deverá ser uma equipa (até de elemento único) com total conhecimento da norma IEC 62303:2006, que, no final de cada *sprint*, realiza uma auditoria às atividades realizadas no decorrer da *sprint*, recolhendo as evidências necessárias para garantir a *compliance* com a norma em questão. Esta equipa deverá comunicar com o *Product Owner* de forma a garantir que qualquer não conformidade é resolvida na iteração seguinte.

5.4 Práticas de Desenvolvimento Propostas

Para além da sequência das atividades no ciclo de vida e das responsabilidades dos elementos da equipa, também há estratégias e ferramentas que a equipa utiliza para desenvolver e que fazem parte da especificação do modelo uma vez que, sem as mesmas, a própria forma como se ligam as atividades e tarefas deixa de fazer sentido.

Nunca é demais reforçar que se pretende que esta metodologia de desenvolvimento, apesar de pretender ser *compliant* com a norma IEC 62304:2006, tem como finalidade estimular e não desvirtuar as filosofias ágeis de desenvolvimento. Assim sendo, as estratégias que se propõe vão de encontro àquelas que são recomendadas pelas equipa *agile*, apesar de não diretamente transpostas no Manifesto [95].

- ***Test Driven Development (TDD)*** [96] é uma abordagem para o processo de codificação que incentiva os programadores a escrever testes para o seu código antes de escrever o código por si só. Neste estilo de desenvolvimento, o programador deve começar por escrever um único teste unitário descrevendo um aspeto do programa, sendo que este teste falhará uma vez que a *feature* está em falta. Depois, deverá escrever o código na sua forma mais simplificada possível para fazer o teste passar e, de seguida, efetuar *refactoring* ao código até que este satisfaça os padrões de qualidade desejados.

- ***Refactoring*** é uma prática que consiste no melhoramento da estrutura interna do código existente, preservando o seu comportamento funcional. Este conceito está bastante relacionado com TDD. Estes dois conceitos estão intimamente ligados com o *agile* uma vez que tentam fornecer um «antídoto» ao problema da crescente complexidade dos projetos. Estas ferramentas permitem modificar constantemente o código em resposta à mudança.

- **A automação do processo de compilação** é também fundamental para auxiliar, não só o processo de desenvolvimento, mas também o processo de gestão de configurações. O objetivo desta prática é reduzir o processo de compilação de código a uma atividade de um único passo que qualquer

elemento da equipa pode efetuar. Isto traz vários benefícios. A título de exemplo, ajuda os programadores a verificar a correção dos pressupostos que fazem enquanto codificam e protege contra eventuais incompatibilidades consequentes da integração de código por parte de outros programadores.

- O próximo passo lógico é um **processo de *deployment*[§] automático**, que ajuda a promover e simplificar a transição de novas versões do sistema para ambientes de teste e para ambientes de produção. Desta forma, facilita-se a execução de testes unitários e de sistema em ambientes equiparáveis com o ambiente onde o *software* irá executar numa fase final e evitam-se problemas de integração ou *performance* com bastante antecedência.

- **Automação de testes** é o primeiro passo que o projeto pode executar no sentido de saber qual é o estado do projeto em determinado momento. Sendo assim, devem ser escritos de forma a acompanhar qualquer novo desenvolvimento ou modificação de código antigo. No nível unitário, os testes devem ser uma extensão do processo de compilação e, desta forma, um teste falhado deve ser encarado da mesma forma que uma *build* falhada. Os testes ao nível de sistema também deverão ser escritos e automatizados.

- A **Integração Contínua** liga o uso de um sistema de controlo de versões com a automação do processo de compilação e teste de forma a fornecer à equipa um nível elevado de confiança relativamente ao correto funcionamento do sistema de *software*. Num ambiente de integração contínua, os programadores podem escrever e submeter código, despoletando sistemas automáticos que compilam, testam e instalam periodicamente novas versões do sistema. Se, por algum motivo, a compilação ou os testes falharem, a equipa é alertada para que a situação possa ser corrigida.

[§] O termo *deployment* em *software* representa todas as atividades que tornam o produto de *software* pronto a utilizar. Inclui instalação, ativação, versionamento e entrega do produto.

6

Compliance com a norma IEC 62304:2006

No capítulo anterior foi apresentado um modelo de ciclo de vida cujo objetivo é, para além a produtividade do desenvolvimento de *software*, atingir *compliance* com a norma internacional recomendada para este processo, aplicado aos dispositivos médicos. Ainda assim, ainda não foi explicado, ao detalhe, como atingir este objetivo.

Neste capítulo abordar-se-ão, ponto a ponto, os principais tópicos para *compliance* relativamente à norma IEC 62304:2006. Para tal, foram selecionados os conceitos mais difíceis de ultrapassar (nomea-

damente tendo em conta as barreiras às metodologias ágeis apresentadas na bibliografia e discutidas no capítulo 4.1.2), para os quais serão explicadas as estratégias e ferramentas propostas e que deverão ser integradas no seio do ciclo de vida.

O anexo A fornece um resumo das estratégias detalhadas neste capítulo, com correspondência para todas as secções do *standard*.

6.1 Ferramentas Transversais

Antes de iniciar uma análise específica a cada conceito escolhido da norma, é importante referir e explicar que existe um conjunto de ferramentas que se consideram fundamentais para atingir os objetivos propostos e que têm uma natureza transversal ao ciclo de vida.

Estas ferramentas permitem, fundamentalmente, criar evidências, evidências estas que, num eventual processo de aprovação, serão os itens procurados para demonstração irrefutável do cumprimento com determinado critério.

- Os sistemas de **controlo de versões** (SCV) são ferramentas fundamentais para o desenvolvimento eficaz de *software*, uma vez que permitem gerir mudanças ao código fonte. As equipas devem utilizar funcionalidades de versionamento, *rollback*, *tagging* e assistência para *merges* [*] fornecidas por estas mesmas. Desta forma, o mecanismo de controlo de versões terá a capacidade de guardar um histórico fiável para todas as linhas de código do projeto [97]. Ainda assim, a equipa de desenvolvimento deverá garantir que faz *commits* [†] diários do código desenvolvido de forma a garantir um histórico o mais granular possível.
  Exemplos: Git [98], Apache Subversion [99]

- As **plataformas de *issue tracking*** (PIT) são, na sua essência, ferramentas que permitem à equipa de desenvolvimento registar e acompanhar o estado de todos os elementos (*issues*) associados com itens de configuração. O *issue tracker* serve como uma base de dados centralizada para registo de anomalias, requisitos (*user stories*), tarefas ou qualquer outra coisa e permitem controlar o seu progresso, modificar o seu estado (aberto, fechado, em desenvolvimento) e atribuir prioridades. Adicionalmente, estas ferramentas, devido à sua imensa versatilidade e facilidade de uso, ganharam uma nova dimensão, nomeadamente como repositório de conhecimento (substituindo *wikis* e blogs) e como canal de comunicação e coordenação entre a equipa e os diversos *stakeholders* e clientes [100].
  Exemplos: JIRA [101], Bugzilla [102]

[*] *rollback*, *tag* e *merge* são conceitos relacionados com o SCV. O primeiro transmite a capacidade de reverter o estado de código até um determinado ponto definido; o segundo, significa a atribuição de um rótulo a um estado definido do código para que possa ser identificado; o terceiro significa uma ação de conciliar mudanças distintas no código.

[†] *commit*, no contexto de um SCV, significa uma submissão de uma nova versão de código no repositório.

- Os **ambientes integrados de desenvolvimento** ou IDE do inglês *integrated development environment* são ferramentas desenhadas para maximizar a produtividade dos programadores, fornecendo um ambiente comum para todas as tarefas de desenvolvimento. A maioria dos IDEs fornecem funcionalidade para alterar, compilar, instalar e fazer *debug* [‡] de *software*, assim como rápida integração com o SCV.
  Exemplos: Visual Studio [103], IntelliJ [104]

6.2 Análise e Gestão de Requisitos

A importância de um conjunto de requisitos prioritizado e que seja entendido por toda a equipa é evidente. Contudo, uma tentativa de definir um agregado completo de requisitos demasiado cedo no ciclo de vida do projeto pode ser contraproducente, restritivo e desadequado para contextos em que o âmbito é variável. Novos requisitos e novas oportunidades podem facilmente apresentar-se e pretende-se que a equipa de engenharia consiga acomodar estas eventualidades.

A forma mais popular de expressar requisitos nos modelos ágeis são as *user stories*, que por sua vez constituem requisitos expressados sob a perspetiva do utilizador final. O formato de uma *user story* é bem definido:

As a <papel ou responsabilidade>
I need <requisito>
So that <objetivo>

As *user stories* fornecem uma mensagem poderosa de que a análise de requisitos tem intenção de trabalhar colaborativamente com os utilizadores e *stakeholders*. Devem ser breves e servir de estímulo para uma discussão mais aprofundada posteriormente.

O detalhe de cada *user story* deve ser documentado em cada *sprint* num Documento de Especificação de Requisitos (DER). Os requisitos podem ser funcionais, de sistema, de segurança, entre outros (conforme definido pela alínea 5.2.2 da norma IEC 62304:2006).

De uma forma simplificada, pretende-se que o ciclo de vida dos requisitos de *software* seja o seguinte:

1. O *Product Owner* tem a responsabilidade de proceder ao levantamento de requisitos junto com os *stakeholders* relevantes. Os requisitos iniciais são obtidos na fase de *product vision*, mas podem surgir novos requisitos em qualquer fase do projeto, através de pedidos de modificação ou de anomalias.

[‡] *debug* é um processo rotineiro de localizar e remover anomalias num programa de *software* através da utilização de ferramentas específicas para o efeito.

2. O requisito é expresso sob a forma de uma *user story* ou *epic* e colocado no *backlog* do produto com uma determinada ordenação de acordo com a sua prioridade. Todas as *user stories* devem ser adicionadas na plataforma de gestão do projeto.

3. Na reunião de *sprint planning*, os requisitos são discutidos com os intervenientes da *sprint*, sendo que alguns integram o *backlog* da *sprint*.

4. Ainda na reunião de *sprint planning*, são avaliados os riscos para as *stories* que integram a *sprint* e reavaliados os riscos prévios pertinentes. São discutidas e documentadas na plataforma de gestão do projeto medidas de controlo de risco para estes itens.

5. Aquando da conclusão de cada *user story*, o DER é atualizado com o detalhe necessário (secção 6.3).

6. Paralelamente à *code review*, o documento modificado deverá ser alvo de verificação (secção 6.4).

6.3 Critérios de Aceitação

A *Definition of Done* (DoD) é crucial para um funcionamento eficiente da *SCRUM Team*, uma vez que promove a qualidade do trabalho desenvolvido, fornecendo uma definição coerente para quando uma *user story* está terminada.

Uma boa DoD garante que a equipa entrega funcionalidades que estão realmente terminadas, não só em termos de funcionalidade, mas também em termos de qualidade. Além disso, permite focar a equipa de desenvolvimento nas atividades que são realmente valiosas para o processo de desenvolvimento. Neste caso em concreto, é importante que a DoD inclua os seguintes pontos:

- o código foi produzido para a funcionalidade desejada;
- o projeto compilou sem erros;
- foram codificados e executados testes ao nível unitário, de integração e de sistema;
- o código atualizado foi *deployed* nos ambientes de teste sem erros;
- foram concluídas todas as atividades de *refactoring*;
- foram concluídos procedimentos para atualizar a documentação, nomeadamente o DAS, o DPS e o DER.
- foi efetuada uma *code review* ao código, testes e verificados os documentos atualizados (6.4);

- foram corrigidas todas as anomalias identificadas no processo de verificação.

É importante realçar que a natureza genérica da DoD também tem algumas limitações. Nem todas as atividades presentes nesta «*checklist*» são aplicáveis transversalmente às *user stories* do projeto. Cabe à equipa e ao *SCRUM Master* decidir de forma consciente quais as atividades que podem eliminar para cada *story*. Poderá haver, por exemplo, *stories* relacionadas com correção de defeitos que não impliquem qualquer atualização de documentação.

Contrariamente à *definiton of Done*, os critérios de aceitação são específicos para cada *user story* e fazem parte do detalhe da mesma. Estes critérios poderão ser escritos pelo *Product Owner* como parte do aperfeiçoamento do *backlog*, e mesmo pela equipa como consequência da análise feita na cerimónia de planeamento da *sprint*.

Os objetivos dos critérios de aceitação são de clarificar com detalhe aquilo que é necessário implementar na *story*, assegurar que todos têm um entendimento comum do problema, ajudar à verificação por parte de colegas e mesmo à codificação de testes e também ajudar na atualização do documento de especificação de requisitos.

6.4 Verificação

Em *software*, a verificação tem a finalidade de determinar se a solução está a ser construída da melhor forma, com vista a satisfazer os requisitos. Tal como na maioria das recomendações para sistemas críticos, a norma IEC 62304:2006 é bastante exigente neste âmbito, impondo a execução atividades de verificação para os processos de implementação de unidades de *software*, testes, requisitos, arquitetura, desenho detalhado, análise de risco e resolução de problemas.

A revisão de código (*code review*) é uma prática fundamental no processo de verificação e enquadra-se na filosofia *agile*. Para além de ajudar a diminuir o risco associado a cada nova instalação, a prática de *code review* apresenta outras vantagens, como a partilha de conhecimento entre elementos da equipa.

Idealmente, deverá ser realizada uma *code review* em cada submissão que é executada no sistema de controlo de versões. Ou seja, se um programador deseja integrar novo código no projeto, este terá de ser verificado por um número pré-determinado de colegas antes que isso possa acontecer.

Uma excelente forma de atingir este objetivo facilmente é através do uso da funcionalidade de *pull request*, disponível na maioria dos sistemas distribuídos de controlo de versões. Usando esta estratégia, quando um programador deseja desenvolver uma nova funcionalidade, cria uma cópia do código fonte do projeto sob a forma de um *branch*. Após efetuar modificações, assinala que pretende integrar o código introduzido no *branch* no repositório centralizado - *pull request* - e apenas após uma verificação e aprovação pode fazer a integração através de um *merge* no repositório central. Este fluxo encontra-se representado na figura 6.1.

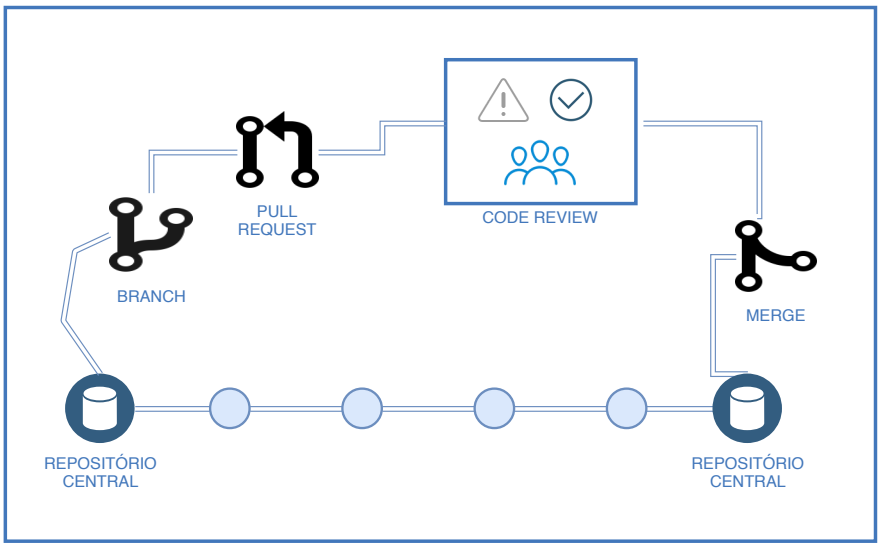


Figura 6.1: Fluxo exemplificativo do trabalho a partir de *branches* e *pull requests* como mecanismo de verificação de código.

Outra grande vantagem deste mecanismo é a produção de evidências de verificação de uma forma automática. Um exemplo é a ferramenta *Crucible* [105] que faz integração automática com A PIT *JIRA* [101], permitindo registar automaticamente todos os comentários, anomalias e verificações efetuadas.

Além disso, estas revisões são úteis para a verificação de detalhes de arquitetura e desenho detalhado, que são expressos por forma de código no decorrer de cada *sprint*. Relativamente aos testes, pretende-se que estes sejam automatizados e, assim sendo, serão submetidos a este processo da mesma forma que o código fonte.

Este mecanismo de revisão de código pode também ser aplicado à revisão de documentos, se os mesmos estiverem incluídos no sistema de controlo de versões, ou seja, quando um documento é atualizado, este é submetido no repositório central e será sujeito a uma revisão. Desta forma, garante-se uma verificação rigorosa no decorrer das *sprints* aos documentos DES, DAS e DPS*Software*.

Ainda assim, a verificação do conteúdo presente no repositório central do sistema de controlo de versões não é suficiente. Numa fase prematura do projeto, pretende-se que a *sprint 0* seja uma estratégia de verificação para toda a documentação e decisões efetuadas no decorrer da *product vision*. Adicionalmente, a cerimónia de *sprint planning* deverá ser uma altura importante para a verificação de medidas de controlo de risco para cada item do *product backlog* por parte de toda a *SCRUM Team*.

6.5 Desenho Detalhado

De acordo com as filosofias ágeis, o desenho detalhado deve emergir no decorrer das *sprints* e nunca ser decidido numa fase prévia. O sistema, de uma forma global, irá evoluir com o tempo para satisfazer novos requisitos e tirar partido de novas tecnologias de uma forma iterativa. Apesar de, nas fases de *product vision* e *sprint 0*, ser definida uma arquitetura genérica para os principais componentes do sistema, pretende-se que esta modelação seja apenas o suficiente para que o projeto se consiga sustentar.

Utilizando a prática de TDD, é possível construir documentação de desenho detalhado sob a forma de testes unitários, que acabam por representar um conjunto de especificações executáveis.

É importante salientar que a natureza iterativa do ciclo de vida também se aplica ao desenho detalhado. Cada membro da equipa, ao abordar a implementação de um conjunto de requisitos deverá sentir a liberdade de analisar, desenhar, codificar e testar, iterando entre atividades como achar pertinente. A verificação do desenho detalhado deverá ser efetuada através de *code reviews* e revisões de documentação, assim como pela execução de testes automáticos.

6.6 Gestão de Configurações

A gestão de configurações (GC) é uma disciplina reconhecida em engenharia que visa fornecer processos e estratégias para identificar e controlar itens, de forma a garantir a integridade e qualidade do produto que se encontra a ser desenvolvido [106]. O conceito de GC pode ser difícil de explicar, mas ainda assim é consensual que cobre a identificação de itens dentro do sistema e o controlo da sua mudança, assim como a mudança do sistema na sua globalidade.

Uma definição conservadora deste processo pode ser satisfeita com o uso de qualquer plataforma de controlo de versões. Pelo contrário, uma definição mais alargada tende a incluir todos os artefactos produzidos no decorrer do projeto, tal como a própria equipa de desenvolvimento e as atividades realizadas pela mesma [97].

A principal estratégia a ter em conta para uma implementação eficiente de um sistema de GC num meio *agile* é a automação de processos, incluindo uma forma de documentar atividades automaticamente. Ainda assim, não deve ser esquecido que os princípios básicos da GC como identificação, controlo, auditoria e registo de itens são agnósticos ao ciclo de vida de desenvolvimento [106].

O SCV é uma das principais ferramentas para gestão de configurações, uma vez que permite que a equipa consiga encontrar sempre o estado do sistema em cada momento da sua existência. É importante acrescentar que esta lógica é aplicável não só ao código fonte da solução, mas também a todos os *scripts* de compilação, instalação e teste, assim como à documentação e outros artefactos. Desta forma, é sempre possível recuar até um estado coerente do projeto e reproduzir todos os passos que levaram a determinada

versão.

A automação do processo de compilação, assim como de *deployment* é também fundamental para o sucesso do sistema de controlo de versões, assim como discutido na secção 5.4. Um mecanismo eficiente de integração contínua pretende aliar todas estas estratégias, com o intuito de fornecer ciclos mais rápidos e maior qualidade da solução final, bem como do processo.

A abordagem *agile* ao processo de gestão de configurações fornece uma solução mais flexível aos problemas, nomeadamente quando existem mudanças de âmbito e de vontades por parte dos *stakeholders*. Adicionalmente, não impede a criação de evidências, uma vez que o sistema de controlo de versões e a plataforma de *issue tracking* garantem registo e rastreabilidade para a maioria dos itens de configuração. Todos os itens de configuração que não podem ser versionados desta forma automática (tais como alguns documentos, *hardware* ou infraestrutura) devem ser introduzidos num documento apelidado Lista de Itens de Configuração (LIC).

6.7 Rastreabilidade

A rastreabilidade consiste simplesmente no potencial para relacionar dados que estejam armazenados em artefactos de algum tipo, juntamente com a capacidade de examinar este relacionamento. A capacidade para conseguir rastreabilidade depende, então, da criação de *links* navegáveis entre artefactos ou dados que, de outra forma, estariam completamente desconectados [107].

Esta capacidade é especialmente importante em *software* considerado crítico, onde a segurança é uma prioridade. A norma IEC 62304:2006 exige rastreabilidade entre diversos artefactos, nomeadamente no que diz respeito às medidas de controlo de risco e aos itens de gestão de configurações (secção 6.6). Isto porque a rastreabilidade providencia visibilidade para diversos aspetos diferentes do sistema a ser desenvolvido e contribui para um melhor entendimento do *software* na sua globalidade.

Em ciclos de vida *agile*, as vantagens da rastreabilidade não são diferentes dos restantes. A análise do impacto de mudanças é facilitada, a necessidade de *compliance* é atingida e é garantido que a funcionalidade desenvolvida vai de encontro aos requisitos existentes.

Uma estratégia eficiente de rastreabilidade raramente acontece devido ao acaso ou por esforços esporádicos e manuais. Principalmente quando o ciclo de desenvolvimento é ágil, ou seja, orientado às pessoas e ao desenvolvimento de funcionalidade, pretende-se que os artefactos sejam, tanto quanto possível, conectados automaticamente entre si no decorrer do processo normal de desenvolvimento ao longo de uma *sprint*.

Se a equipa de desenvolvimento trabalhar com o nível de granularidade que a estratégia de TDD e a automação de testes dita, então atividades como codificação, desenho e teste fazem, fundamentalmente, parte da mesma tarefa. Conectar estas atividades ao mesmo identificador no SCV permite atingir rastrea-

bilidade entre estes artefactos. A rastreabilidade é extremamente dificultada quando estas atividades estão completamente desacopladas e pertencem a fases diferentes do ciclo de vida.

É também importante centralizar a informação sobre os artefactos e relacioná-la com a pessoa responsável pela sua criação. Uma *wiki* do projeto ou uma PIT são exemplos de ferramentas que ajudam a resolver esta questão.

Figura 6.2: Ilustração dos vários elementos que podem ter rastreabilidade a partir do sistema de controlo de versões e da plataforma de *issue tracking* do projeto.

Mas, mais do que tudo, é fulcral atingir automação das tarefas morosas e desgastantes relacionadas com a rastreabilidade. Uma via para atingir este objetivo é através da integração entre o IDE, o sistema de controlo de versões e a plataforma de *issue tracking*. Estas ferramentas são essenciais para proporcionar transparência ao processo e ao ciclo de vida [108]. Se, por exemplo, uma submissão de código no sistema de controlo de versões for associada com o identificador de uma *user story* no sistema de controlo de versões, é possível atingir rastreabilidade entre código, testes, requisitos e também itens de gestão de configurações, se for o caso.

Ainda assim, existem alguns passos manuais que são necessários, neste caso em particular, numa vez que é necessária a produção de alguma documentação. Nos documento de especificação de requisitos, arquitetura e planeamento de *software,* deve ser indicado o identificador de cada *user story* na PIT.

Adicionalmente, nos itens criados que dizem respeito a riscos, devem ser introduzidas as ligações apropriadas para as *user stories*, garantindo rastreabilidade para os requisitos e submissões no sistema de controlo de versões.

6.8 Teste

Testar o *software* desenvolvido é indispensável pelo simples facto de que erros acontecem. Quando estamos perante um sistema crítico como um dispositivo médico, estes erros tornam-se potencialmente perigosos e o risco é mais elevado. Num ambiente de TDD e Integração Contínua, como aquele que é sugerido, pretende-se que haja um elevado grau de automação dos testes. Cada submissão no sistema de controlo de versões poderá potencialmente desencadear um processo de compilação, que apenas será bem sucedido se todos os testes executarem com sucesso.

O *standard* IEC 62304:2006 exige evidências de testes a três níveis distintos:

1. **integração**. Através de ferramentas de virtualização, é possível obter testes compreensivos de integração de componentes no próprio processo de compilação [109].

2. **sistema e regressão**. É possível também automatizar testes ao nível do sistema através de ferramentas dedicadas a esse fim, como *Selenium* [110]. Construir testes automatizados garante que podem ser executados a qualquer altura, facilitando a execução de testes de regressão.

As ferramentas de teste, quer sejam executadas no processo de compilação ou não, permitem geração automática de relatórios detalhados da sua execução, o que é importante para garantir evidências da sua execução.

6.9 Gestão de Mudanças e Resolução de Problemas

A gestão e análise de potenciais mudanças às especificações originais é um processo bastante facilitado pela natureza iterativa do ciclo de vida ágil. Qualquer mudança de âmbito deve ser identificada e aprovada pelo *Product Owner* e convertida em itens introduzidos no *backlog* do produto com uma prioridade associada. Estes itens darão origem a *user stories* e serão analisados como qualquer outro item na reunião de planeamento da *sprint*.

O mesmo mecanismo é aplicável relativamente à resolução de eventuais anomalias no produto. Estes defeitos deverão ser introduzidos como itens de configuração na PIT do projeto, devidamente assinalados como defeitos ou *bugs*. Estes itens deverão ser alvo de análise, também, nas reuniões de planeamento das *sprints*. Adicionalmente, o *SCRUM Master* tem a responsabilidade de analisar tendências relativamente às anomalias identificadas e identificar medidas caso se justifique.

6.10 *Release*

A entrega ou instalação de uma nova versão do produto de *software*, ou a *release*, pode ser considerada um forte barómetro da agilidade da equipa de desenvolvimento. Todo o esforço para desenvolver, planear e

testar rapidamente pode ser em vão se o processo de entrega não resultar. Para tornar o processo de *release* eficiente, automação é chave, assim como um processo eficiente de integração contínua.

É também fundamental estimular ativamente um ambiente de transparência em cada entrega. Em primeiro lugar, as versões que foram entregues devem ser devidamente identificadas no SCV (por exemplo, usando *tags*). Adicionalmente, deve ser criado um item de configuração na PIT do projeto onde devem ser identificados os defeitos conhecidos e a forma como foi feita a instalação.

Em suma, as estratégias apresentadas têm o fim de permitir criar evidências transparentes de cumprimento dos requisitos da norma IEC 62304:2006, ao mesmo tempo que têm características com as quais os programadores se identificam e que não trazem *overhead* para as suas tarefas diárias, permitindo que a *sprint* aconteça de uma forma natural. Para tal, é fundamental a automação (de testes, de *builds*, de *deployments*), a utilização de uma PIT e de um sistema de VCS. O Anexo D apresenta uma *fact-sheet* que sumariza o modelo de ciclo de vida apresentado, bem como os conceitos principais para atingir *compliance* com a norma supramencionada.

Parte III

Validação da Solução

7
Validação do Modelo Proposto

Neste capítulo apresentam-se os esforços efetuados no sentido de obter uma validação da metodologia proposta. Os objetivos principais do ciclo de desenvolvimento apresentado são os de permitir *compliance* com a norma IEC 62304:2006 de uma forma direta, bem como não desvirtuar as filosofias subjacentes à utilização do *SCRUM*.

Idealmente, o ciclo de vida seria implementado numa equipa real, no decorrer do desenvolvimento de uma aplicação potencialmente classificada como um dispositivo médico, com uma classificação de segurança superior à classe A (de forma a possibilitar o teste a todas as alíneas presentes na norma). Esta dissertação foi efetuada paralelamente com uma atividade profissional em tempo completo por parte do

autor. Devido a estas limitações de calendário e recursos, não foi possível reunir as condições necessárias para proceder à validação da metodologia nestes moldes. Como alternativa, realizaram-se dois tipos de experiências, que constituem o conteúdo das próximas secções.

7.1 Validação em Ambiente Académico

7.1.1 Apresentação da Abordagem

No âmbito da unidade curricular de Laboratório de Gestão de Projetos (LGP), que é lecionada ao longo do segundo semestre de vários cursos diferentes na área da engenharia da Faculdade de Engenharia da Universidade do Porto, os estudantes são convidados a desenvolver projetos de intervenção em contexto real para empresas parceiras da mesma instituição de ensino superior.

LGP tem um enquadramento de catorze semanas organizadas em várias etapas, tal como refletido na tabela 7.1.

Tabela 7.1: Calendário das atividades da unidade curricular de LGP

Fase	Datas
Início	9 de fevereiro a 16 de fevereiro de 2018
Conceção	23 de fevereiro a 23 de março de 2018
Desenvolvimento	6 de abril a 25 de maio de 2018
Encerramento	22 de junho de 2018

Surgiu a oportunidade de colaborar com uma destas equipas, cujo projeto, apelidado de *Prehab*, cumpria os requisitos para ser classificado potencialmente como dispositivo médico. Encomendado pelo Hospital de São João, o *Prehab* apresentou-se como uma plataforma que pretendia dar aos médicos a habilidade de monitorizar e controlar o estado de saúde de pacientes com doença hepatobiliar, no período que antecede a operação. O principal objetivo foi desenvolver um programa de «Reabilitação Multimodal Peri-Operativa», baseado em atitudes protocoladas suportadas por evidências científicas, com foco na recuperação funcional dos pacientes, redução da mortalidade e do tempo de internamento.

Esta aplicação, com componente móvel e *web*, enquadrou-se no conjunto de finalidades previstas aceites pelo RDM e foi classificado como Classe I pela equipa de desenvolvimento (Classificação de segurança A nos temos da norma IEC 62304:2006).

A equipa de desenvolvimento era constituída por nove estudantes do primeiro ano do Mestrado em Engenharia de *Software*, um do Mestrado em Multimédia e um do Mestrado em Engenharia de Gestão de Serviços, sendo que um deles desempenhou a função de líder de equipa. Os estudantes não tiveram

alocação completa a este projeto, uma vez que o currículo dos mestrados inclui mais quatro unidades curriculares no segundo semestre do primeiro ano. Adicionalmente, a maioria dos estudantes também desempenha atividades profissionais paralelamente aos estudos.

O objetivo desta abordagem foi de introduzir o ciclo de vida proposto aos estudantes integrantes da equipa *Prehab* e estudar a sua adaptação com o mesmo. A unidade curricular onde a equipa se insere promove, à partida, as metodologias ágeis para o trabalho diário dos estudantes. O papel da *QA Team* foi desempenhado por elementos externos à equipa, nomeadamente pelo autor desta dissertação.

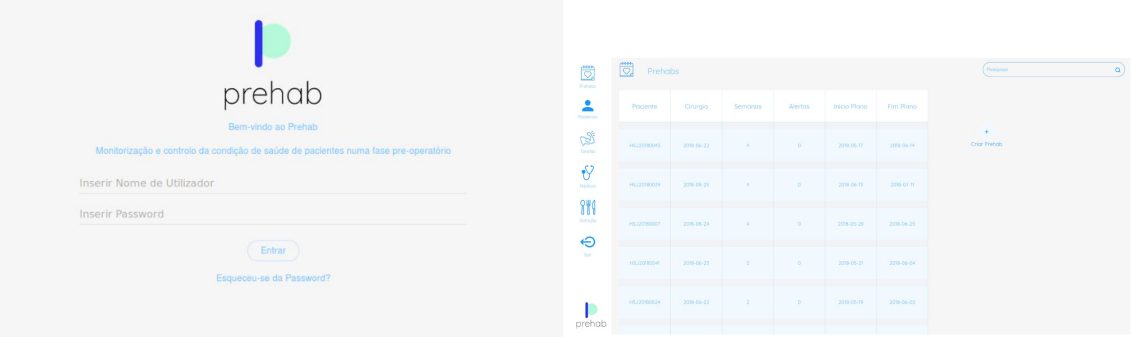


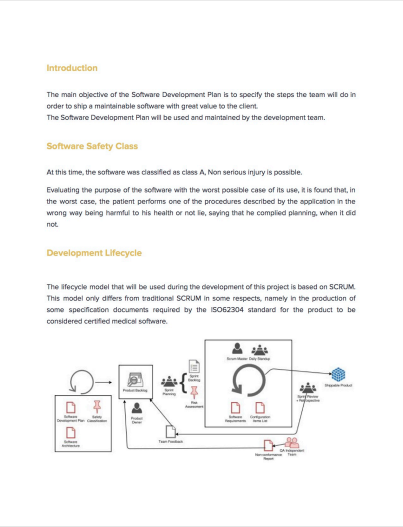
Figura 7.1: Capturas de ecrã da aplicação *Prehab*

7.1.2 Resultados Obtidos

A equipa de desenvolvimento mostrou-se bastante proativa, especialmente numa fase inicial do projeto, o equivalente ao *Product Vision*, fase no decorrer da qual foram capazes de produzir bastantes evidências satisfatórias para o eventual processo de aprovação. Nesta fase, foram produzidos os seguintes *deliverables*:

- Documento de Planeamento de *software*, no qual foi detalhada a classificação de segurança do produto, ciclo de vida de desenvolvimento e planificação para documentação, análise de risco, resolução de problemas, testes e rastreabilidade (imagem à esquerda na figura 7.2);
- Documento de Arquitetura de *Software*;
- Especificação de interfaces gráficas da aplicação.
- Documento de Análise do Risco, onde foram identificados e prioritizados os principais riscos identificados numa fase inicial do projeto (imagem à direita na figura 7.2).

No decorrer das *sprints* de desenvolvimento, porém, houve entraves que dificultaram a obtenção de resultados conclusivos e mensuráveis da abordagem:



Introduction

The main objective of the Software Development Plan is to specify the steps the team will do in order to ship a maintainable software with great value to the client.
The Software Development Plan will be used and maintained by the development team.

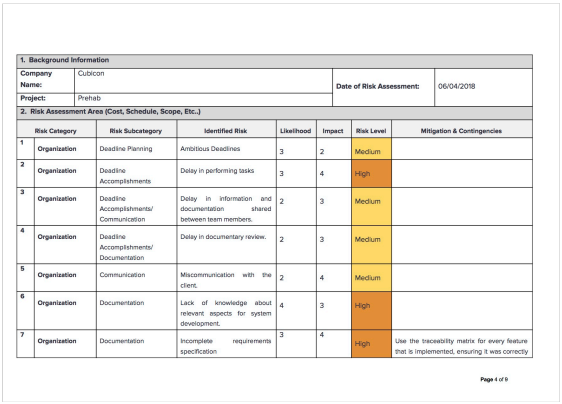
Software Safety Class

At this time, the software was classified as class A, Non serious injury is possible.

Evaluating the purpose of the software with the worst possible case of its use, it is found that, in the worst case, the patient performs one of the procedures described by the application in the wrong way being harmful to his health or not lie, saying that he complied planning, when it did not.

Development Lifecycle

The lifecycle model that will be used during the development of this project is based on SCRUM. This model only differs from traditional SCRUM in some respects, namely in the production of some specification documents required by the ISO62304 standard for the product to be considered certified medical software.



1. Background Information

Company Name:	Cubicon	Date of Risk Assessment:	06/04/2018
Project:	Prehab		

2. Risk Assessment Area (Cost, Schedule, Scope, Etc..)

	Risk Category	Risk Subcategory	Identified Risk	Likelihood	Impact	Risk Level	Mitigation & Contingencies
1	Organization	Deadline Planning	Ambitious Deadlines	3	2	Medium	
2	Organization	Deadline Accomplishments	Delay in performing tasks	3	4	High	
3	Organization	Deadline Accomplishments/ Communication	Delay in information and documentation shared between team members.	2	3	Medium	
4	Organization	Deadline Accomplishments/ Documentation	Delay in documentary review.	2	3	Medium	
5	Organization	Communication	Miscommunication with the client.	2	4	Medium	
6	Organization	Documentation	Lack of knowledge about relevant aspects for system development.	4	3	High	
7	Organization	Documentation	Incomplete requirements specification	3	4	High	Use the traceability matrix for every feature that is implemented, ensuring it was correctly

Page 4 of 9

Figura 7.2: Exemplos dos entregáveis produzidos pela equipa *Prehab*. À esquerda, uma página do Documento de Planeamento de *Software* e, à direita, uma página do Documento de Análise do Risco.

- Não se tratando de uma organização, mas sim de um grupo de estudantes, não foi possível construir e apresentar um sistema de gestão da qualidade, tal como exigido pela norma em questão.
- Tratando-se de um contexto académico, nos quais os estudantes não tem alocação exclusiva ao projeto em questão, tornou-se difícil implementar a maioria das práticas promovidas pelas metodologias ágeis tais como as práticas e desenvolvimento (TDD, *refactoring*), de teste (automação) e também as cerimónias do *SCRUM*;
- Não foi possível encontrar elementos para desempenhar todos os papéis necessários, sendo que a ausência de um *Product Owner* impossibilitou algumas ações importantes no decorrer do processo.

Ainda assim, destacam-se os seguintes aspetos positivos:

- A equipa de desenvolvimento implementou com sucesso alguns mecanismos de gestão de configurações, nomeadamente a partir da utilização de um *issue tracker* (*Pivotal Tracker* [111]) e de um sistema de controlo de versões (*Git* [112]), mas também com a implementação de uma *pipeline* de integração contínua (*Travis* [113]);
- Houve atualização de documentos no decorrer das *sprints*, no que diz respeito a requisitos, arquitetura e risco;
- A equipa mostrou capacidade de resolver problemas, quando era apresentado um item de alta prioridade no *backlog*, concluindo a tarefa identificada numa das *sprints* seguintes.

7.2 Validação por Entrevista Guiada

7.2.1 Apresentação da Abordagem

As entrevistas guiadas realizadas no âmbito desta dissertação tiveram, como principais objetivos:

- Validar a pertinência das temáticas estudadas na dissertação, nomeadamente a necessidade e possibilidade de introduzir práticas ágeis no desenvolvimento de software crítico;
- Validar as contribuições efetuadas, no que diz respeito às alterações introduzidas ao ciclo de desenvolvimento *SCRUM*;
- Identificar, junto de profissionais credenciados na área da engenharia de *software*, pontos de melhoria e possíveis oportunidades para trabalho futuro;
- Aprofundar o conhecimento relativamente ao ecossistema legal dos Dispositivos Médicos em Portugal e na União Europeia;
- Compreender os desafios presentes e futuros que a indústria dos dispositivos médicos enfrentam em Portugal e na União Europeia, quer do ponto de vista dos fabricantes, quer do dos avaliadores.

7.2.2 Entrevistas a profissionais na área da Engenharia

Dentro do leque de possíveis perfis para candidatos à realização destas entrevistas, considerou- se fulcral a experiência profissional na área do desenvolvimento de sistemas críticos, sujeitos a fortes restrições relacionadas com o cumprimento de regulamentações e de processos de qualidade.

A *Critical Software* é uma empresa portuguesa especializada no desenvolvimento de soluções de software e serviços de engenharia de informação para o suporte de sistemas críticos orientados à segurança, à missão e ao negócio de empresas [114]. Foram selecionados profissionais desta empresa, com experiência a lidar com metodologias ágeis no contexto do desenvolvimento de sistemas críticos.

Além disso, de forma a conseguir atingir os principais objetivos desta série de entrevistas, julgou-se essencial selecionar perfis que retratassem a pluridisciplinaridade da prática da engenharia de software, recrutando elementos com perfis variados. Sendo assim, foram selecionados os indivíduos espelhados pela tabela 7.2.

As entrevistas foram conduzidas presencialmente nos escritórios da *Critical Software* no Porto, em Portugal, à exceção da entrevista ao Vítor Vieira, que por motivos geográficos respondeu às questões através de vídeo-chamada. Um guião detalhado das mesmas pode ser encontrado na tabela C.1 do Anexo C.

Tabela 7.2: Descrição dos intervenientes das entrevistas guiadas.

Nome	Descrição	Data da Entrevista
Patrick Machado	*Technical Manager* na *Critical Software*; Certificação como *SCRUM Master*; Experiência no desenvolvimento de sistemas críticos (indústrias financeira, justiça) e liderança de equipas;	18 de maio de 2018
Patrício Correia	*Senior Engineer* na *Critical Software*; Certificação como *SCRUM Master*; Experiência no desenvolvimento de sistemas críticos (indústrias financeira, seguros) e liderança de equipas;	23 de maio de 2018
Mónica Sobreira	*Project Manager* na *Critical Software*; Experiência na gestão de projetos críticos (indústrias financeira, telecomunicações, saúde e justiça);	28 de maio de 2018
Vítor Vieira	CEO na Inova DE GmbH Experiência com desenvolvimento de projetos de *software* no setor dos dispositivos médicos	28 de maio de 2018

A entrevista foi estruturada por tópicos, muitos dos quais transversais a todos os entrevistados, mas também existiam alguns tópicos específicos, mediante as características profissionais de cada um dos intervenientes. No Anexo C podem também ser encontrados todos estes tópicos, assim como as questões que foram feitas para cada um deles.

Pretendia-se que os entrevistados falassem abertamente sobre cada um dos temas, descrevendo o que achassem pertinente e falando da sua experiência pessoal e profissional relativamente a cada tópico. Antes da entrevista, foi partilhado com os entrevistados um artigo científico redigido no âmbito desta dissertação, que detalha os objetivos da mesma, assim como o ciclo de vida proposto.

Barreiras ao Desenvolvimento Ágil

O principal objetivo deste tópico foi a comparação entre aquilo que foi discutido no capítulo 4, como sendo as principais barreiras para a implementação de metodologias ágeis para o desenvolvimento de sistemas em ambiente regulamentado, com a experiência profissional de cada um dos entrevistados. De salientar que nenhum dos entrevistados trabalhava, à data da realização das entrevistas, com ciclos de vida puramente ágeis.

O Patrick Machado, em primeiro lugar, reforçou que as características e imposições do cliente podem ser impeditivas para levar a cabo algumas das práticas advogadas pelo *agile*, tais como o *deployment* contínuo: «Estes clientes tendem a contratar serviços de infraestrutura mais tradicionais e retiram isso do

âmbito. Estratégias como *continuous delivery* vão bater com a última parte do processo.». Mencionou também o risco associado a cada *release*, quando se fala em sistemas críticos também é bastante mais elevado, levando o cliente a não se comprometer com ciclos de vida ágeis, não participando ativamente no mesmo. Outro aspeto discutido foi o da evolução dos processos de auditoria, que, na opinião do entrevistado, é bastante mais lento do que dos processos de desenvolvimento. Isto provoca que muitas práticas de desenvolvimento mais avançadas (e promovidas pelos modelos ágeis) não sejam reconhecidas como evidências aceitáveis para alguns processos de certificação. Por exemplo, as cerimónias ágeis fornecem, por si, mecanismos de *inspect and adapt*, contudo, estes mecanismos não satisfazem o grau de formalidade exigido pela maioria das auditorias de qualidade que necessitam de evidências mais formais.

O Patrício Correia, falou, em primeiro lugar, da dificuldade da produção de documentação formal no decorrer da *sprints* ágeis. Além disso, referiu que o âmbito dos sistemas críticos é normalmente mais limitado e bem definido: «neste tipo de sistemas não costumas ter um *scope* aberto», dificultando a priorização de requisitos. Terminou o tópico acrescentando que, contrariamente àquilo que referiu o Patrick Machado, não era da opinião de que a estratégia de integração e *deployment* contínuo corressem o risco de ser infrutíferos uma vez que a obtenção de um produto entregável não significa obrigatoriamente que tenha de haver uma *release* num ambiente produtivo.

A Mónica Sobreira também referiu aspetos relacionados com a natureza dos clientes característicos dos sistemas críticos, mas focou, também, nas características das organizações que estão habituadas a trabalhar de uma determinada forma orientada à planificação: «As organizações estão habituadas a trabalhar de determinada forma. É uma zona de conforto.». Referiu também que os clientes não aceitam as metodologias ágeis por desconhecimento e falta de informação, encarando o *agile* como uma forma de alterar âmbito de uma forma desorganizada, sem comprometimento com decisões.

Finalmente, o Vítor Vieira apontou a rastreabilidade como a principal barreira que tem impedido as equipas que desenvolvem *software* associado a dispositivos médicos, assim como a produção de evidências esclarecedoras para todos os pontos necessários. Além disso, referiu que a implementação de modificações (*change requests*) é bastante difícil em projetos deste tipo, uma vez que exigem um grau de formalidade bastante elevado que inclui aprovações e evidências das mesmas.

Benefícios das Práticas Ágeis

O Vítor Vieira continuou dizendo que as práticas *agile* são, de facto, de grande vantagem com projetos que já se encontram em funcionamento em ambientes produtivos (manutenção de sistemas), uma vez que promovem a rápida implementação de modificações. A iteratividade, neste caso, apresenta grandes benefícios. Reforçou, ainda assim, que na sua experiência o *V-Model* tradicional apresenta mais vantagens para projetos construídos de raiz.

A natureza iterativa dos ciclos de vida ágil foi algo referido por todos os entrevistados como uma grande vantagem do *agile*. O Patrick Machado reforçou a natureza dos ciclos de vida como o *waterfall*, onde «o sistema tem de ser inventado todo na sua fase inicial», a equipa tem de ser capaz de tomar decisões e comprometer-se com preços e *timings*, ao contrário do *SCRUM*, onde cada iteração é influenciada pelo resultado da anterior. O mesmo entrevistado enfatizou também a ideia de que, quando o ciclo de vida não é ágil e o cliente não compreende esta filosofia, é gasta muita energia (de ambos os lados) a contratualizar o âmbito, energia esta que deveria ser despendida a produzir *software*. Esta última ideia também foi discutida pela Mónica Sobreira, que foi da opinião de que existem negociações de âmbito insignificantes que o ciclo de vida pode ajudar a reduzir: «Andas sempre a negociar âmbito.».

Tanto o Patrício Correia como a Mónica Sobreira reforçaram a capacidade que os ciclos de vida ágil têm para criar produtos mais adequados às necessidades e às vontades reais dos *stakeholders*, uma vez que estas podem facilmente mudar no decorrer da execução dos projetos. A Mónica Sobreira acrescentou, porém, que estas vantagens só são possíveis se ambas as partes compreenderem e estiverem sintonizadas relativamente à metodologia de desenvolvimento.

O Patrício Correia concluiu esta ideia, dando um exemplo de um dispositivo médico com uma finalidade prevista de previsão, que pode beneficiar de um ciclo iterativo para melhorar o seu algoritmo de previsão adquirindo gradualmente novos dados da utilização do sistema.

Importância do *tailoring* no desenvolvimento de software

Nenhum dos entrevistados mostrou hesitação em afirmar que o *tailoring* é uma prática fundamental para o sucesso de qualquer projeto de engenharia de *software*. A Mónica Sobreira sublinhou a ideia de que existem muitos projetos (nomeadamente em indústrias como a da aeronáutica ou aeroespacial) em que as metodologias *agile* podem não ser concretizáveis devido às características tecnológicas, mas também porque existem bastantes entidades envolvidas. Contrariamente, poderá haver outros projetos, de outras particularidades, em que as filosofias ágeis se enquadram, sendo que o ideal será sempre encontrar um meio termo que se enquadre no contexto particular de cada projeto. Aliás, acrescentou que modificações no ciclo de vida podem acontecer ao longo dos projetos, caso haja mudanças que o justifiquem.

O Patrício Correia acrescentou, porém, que é importante um conhecimento profundo das práticas standardizadas antes de implementar modificações às mesmas e até mesmo começar a desenvolver utilizando um ciclo de vida estabelecido, de forma não desvirtuar os mesmos. A ideia de não desvirtuar o ciclo de vida original foi também reforçada pelo Patrick Machado, que, excetuando essa ressalva, afirmou que «O *tailoring* é essencial.».

Importância das práticas de *quality assurance* no desenvolvimento de sistemas críticos

O objetivo deste tópico foi, essencialmente, discutir os benefícios do «reverso da medalha» do desenvolvimento ágil. As empresas utilizam práticas de QA como auditorias e revisões para garantir que obedecem às exigências das normas e certificações, práticas estas que também apresentam vantagens para o sucesso dos projetos, na opinião dos entrevistados.

O Vítor Vieira reforçou a importância destas práticas especificamente no contexto dos dispositivos médicos, uma vez que são produtos que, em caso de falha, podem pôr em causa a vida das pessoas. Explicou também que estas práticas são uma consequência da necessidade de ter confiança no correto funcionamento dos produtos. A Mónica Sobreira partilhou desta opinião referindo que «é importante porque são práticas de mitigação de riscos. É impedir que um problema não volte a acontecer». Ainda assim, acrescentou também que as práticas de QA têm de ser sujeitas a *tailoring* para se conseguirem adaptar ao contexto em que se inserem, especialmente quando há possibilidade de automatizar partes do processo. Na opinião da entrevistada, os esforços para automatizar a geração de evidências são fundamentais para reduzir o *overhead* relacionado com processos de certificação e as entidades reguladoras devem ajustar-se a esta realidade.

Já tanto o Patrick Machado como o Patrício Correia referiram que as práticas de QA são importantes para forçar as equipas a adquirir práticas que, de outra forma, seriam negligenciadas, tais como a gestão de configurações e os ciclos de aprendizagem contínua.

Ferramentas

Neste segmento da entrevista, foi pedido aos entrevistados para enumerarem as principais ferramentas (de um ponto de vista da tecnologia) que consideram decisivas para ajudar na criação de evidências em processos de *quality assurance*.

Todos os entrevistados referiram que uma plataforma de *issue tracking* é fundamental neste aspeto. No caso dos profissionais da *Critical Software*, elogiaram a ferramenta JIRA para rastreabilidade, registo de problemas e gestão de risco. «Todas estas evidências devem estar nesta plataforma e não em documentos» referiu Patrick Machado no decorrer esta discussão. Já Vítor Vieira, falou numa experiência positiva com a ferramenta Bugzilla, relativamente à qual destacou a grande vertente de personalização.

O Patrício Correia acrescentou que é importante que as equipas sintam que podem, até, criar as próprias ferramentas de automação auxiliares aos seus ciclos de vida: «Aqui estamos a desenvolver um plugin que permite fazer *upload* dos requisitos a partir de documentos para o JIRA.». Adicionou que a *definition of done* das equipas ágeis deve espelhar as necessidades da equipa, sejam elas testes ou atualização de

documentos.

Gestão do Risco

Este tópico foi abordado apenas com a Mónica Sobreira, de uma perspetiva de gestão. Pretendeu-se saber qual a importância de conceber um processo bem definido para a gestão de risco em particular, assim como as principais ferramentas utilizados neste processo, na experiência profissional da gestora de projetos.

A entrevistada salientou a importância de haver estratégias bem delineadas no sentido de garantir que, quando um problema ocorre, não se vai repetir. Além disso, reforçou que um processo eficiente para mitigar risco nunca depende das ações de uma pessoa, em contrapartida, o processo flui e, por si só, faz parte da forma de trabalhar das equipas (*continuous improvement*): «Tenho de pôr um processo no terreno que não dependa de alguém fazer alguma coisa, tem de surgir naturalmente. O risco é todos os dias».

Em relação à utilização de ferramentas, a Mónica Sobreira referiu o JIRA e também atas de reuniões com os clientes como formas de mostrar evidências da gestão do risco a eventuais auditorias. Foi ainda mais longe, referindo que, anteriormente, na empresa onde trabalha, eram usados apenas documentos de texto para documentar os riscos, o que apresentou graves problemas porque os riscos ficavam «esquecidos» no sistema de controlo de versões. A usabilidade também era um problema, devido à natureza não colaborativa deste tipo de documentos. O JIRA foi uma solução que permitiu colaboração e dinâmica a este processo.

Validação do modelo proposto

Este segmento da entrevista teve como finalidade obter a opinião e a validação dos intervenientes relativamente ao ciclo de vida de desenvolvimento apresentado no capítulo 5 desta dissertação. Foi mostrada a Figura 5.1 aos entrevistados e explicadas as diferentes fases do ciclo de vida, com ênfase nos itens de *tailoring* introduzidos ao modelo *SCRUM*. Para cada um deles, foi pedido aos entrevistados se consideravam a alteração apropriada, se achavam que trazia *overhead* indesejado ao modelo e se achavam suficiente para a produção das evidências necessárias a um eventual processo de auditoria.

Em primeiro lugar, no que diz respeito à *Product Vision* como etapa inicial dos projetos, todos os entrevistados consideraram a sua existência, especialmente quando a criticalidade do sistema é um fator. O Patrício Correia mostrou, porém, alguma reticência relativamente à produção de um documento de arquitetura de *software* numa etapa tão precoce do projeto, argumentando que a duração ideal da *Product Vision* não é compatível com a criação deste documento.

Todos os entrevistados concordaram com a abordagem apresentada relativamente à análise iterativa de risco introduzida na cerimónia de *sprint planning*. Além disso, a Mónica Sobreira acrescentou que talvez fosse boa ideia fazer uma análise do risco adicional na cerimónia de *sprint retrospective*, numa perspetiva de *inspect and adapt*: «Numa reunião de planeamento, pensas em riscos técnicos e funcionais. Deverá haver um momento em que o *Product Owner* e o *SCRUM Master* pensam em conjunto no risco de uma forma transversal.».

De seguida, falando do ciclo iterativo imposto pelas *sprints*, nenhum dos participantes mostrou qualquer preocupação no sentido em que a documentação exigida pudesse pôr em causa a produtividade da equipa. O Patrício Correia voltou a reforçar que a *definition of Done* pode servir para garantir que a documentação era produzida, se necessário, para cada *user story* e o Vítor Vieira reforçou a importância de manter um documento de requisitos sempre atualizado.

Relativamente à produção de documentação no decorrer dos ciclos de desenvolvimento, a Mónica Sobreira voltou a realçar a possibilidade de automatizar este processo de forma a otimizar o trabalho dos programadores e diminuir as tarefas menos estimulantes. Na opinião da mesma, deve ser feito um esforço para separar a linguagem dos gestores de projeto e analistas funcionais, que estão acostumados a trabalhar com documentação mais funcional, da linguagem dos codificadores que estão mais confortáveis a trabalhar com plataformas como o *issue-tracker* ou o IDE. Deu um exemplo interessante de um projeto onde se construíram ferramentas para que os requisitos fossem importados para o JIRA através de um plugin, onde podiam sofrer alterações e exportados novamente para o formato original a qualquer momento.

Já o Patrick Machado mostrou preocupações relativamente à dificuldade de produzir evidências palpáveis de testes rastreáveis para o documento de requisitos apenas através de estratégias como TDD e automação: «Como garantir que os requisitos estão cobertos?».

Falando do tópico que diz respeito à introdução de uma equipa independente de *quality assurance*, como mecanismo de verificação de *compliance* de acordo com as normas, nenhum dos entrevistados teve nada a acrescentar, concordando com a abordagem e achando benéfica para o sucesso do ciclo de vida.

Finalmente, o Vítor Vieira sugeriu a introdução de um novo item de *tailoring* a que apelidou de *market observation* e que deve ter lugar após cada *sprint*, paralelamente ao processo de *release*. No decorrer desta atividade, a equipa deve constatar se, no mercado, houve reclamações e anomalias detetadas em produtos semelhantes. Se pertinente, a equipa deve construir evidências de que estas anomalias não se encontram presentes na solução desenvolvida. Esta atividade é uma exigência do regulamento para os dispositivos médicos.

7.2.3 Entrevistas a profissionais na área dos Dispositivos Médicos - INFARMED

Para além de todas as entrevistas realizadas a profissionais credenciados na área da engenharia e desenvolvimento de sistemas críticos, surgiu a oportunidade de conversar com Mariana Madureira, que pertence à Direção de Produtos de Saúde do INFARMED, entidade competente em Portugal no que diz respeito à avaliação de dispositivos médicos. Esta entrevista foi realizada no dia 7 de Junho de 2018. O guião detalhado desta entrevista pode ser consultado na tabela C.2 do Anexo C.

O primeiro tópico lançado à discussão foi o impacto do novo Regulamento para os Dispositivos Médicos (2017/745/EEC) no panorama europeu e mesmo no nacional. A Mariana Moreira começou por adiantar que considerava o novo regulamento «bastante grande e difícil de digerir», sendo que as autoridades competentes estão neste momento numa fase de implementação do mesmo. As entidades que sofrerão um impacto maior são aquelas que já tem produtos em funcionamento e que serão obrigadas a fazer uma atualização dos seus processos para cumprirem os novos requisitos do RDM.

Contudo, reforçou que o novo regulamento é de «grande utilidade para fabricantes que estejam a começar a desenvolver novos produtos», nomeadamente no mundo do *software*. Este regulamento foi, nas palavras da entrevistada, despoletado por questões tecnológicas relacionadas com as tecnologias da informação e evolução digital, que inevitavelmente se cruzaram com a área dos dispositivos médicos. Sendo assim, este novo conjunto de normas vem trazer mais detalhe para conceitos que estavam subentendidos nas antigas diretivas, assim como introduzir conceitos novos como a interoperabilidade e a cibersegurança. Isto vem ajudar no trabalho, não só dos fabricantes, mas também das autoridades aprovadoras, que já começavam a sentir dificuldades em garantir o cumprimento os requisitos antigos, em contextos mais tecnológicos.

Seguidamente, introduziu-se uma linha de conversação destinada a entender melhor o processo de aprovação de dispositivos médicos, e como funcionaria o mesmo no caso de produtos exclusivamente *software*. A entrevistada referiu que o INFARMED apenas é responsável pela aprovação de dispositivos classificados com classe de segurança I (auto-certificados), sendo que, para os restantes, o processo é conduzido por Organismos Notificados (ON), designadas pelo INFARMED (estes organismos podem ser encontrados em [115]). Nas palavras da mesma:

«No caso do software, enquanto o dispositivo médico pertencer a uma das seguintes classes: classe IIa, IIb ou III - este terá de ser submetido a um processo de avaliação de conformidade junto de um Organismo Notificado (um organismo de avaliação designado e monitorizado pela Autoridade, de nomeação nacional, sendo reconhecido pela Comissão Europeia, que lhe atribui um código de identificação de quatro algarismos. Este código aparece associado à marcação da Comissão Europeia (CE) nos produtos que por ele foram avaliados. Um ON é responsável por efetuar os procedimentos de avaliação de conformidade; autorizar a aposição da marcação CE; emitir os certificados de conformidade; assegurar que o

fabricante cumpre corretamente com as obrigações decorrentes do sistema de qualidade aprovado, etc.)»

Quando questionada sobre o interesse dos fabricantes para o desenvolvimento deste tipo de produtos em Portugal, a Mariana Moreira referiu que «é uma área de grande desenvolvimento em Portugal», não só ao nível dos fabricantes, mas também dos próprios procedimentos de aprovação. Porém, continuou dizendo que considera que há, ainda, um grau assinalável de desconhecimento dos fabricantes relativamente aos procedimentos e à regulamentação específica que precisam de cumprir, sendo que cabe às autoridades a divulgação destes aspetos. Este ponto ganha relevância com a introdução do novo regulamento, que exige o cumprimento de mais requisitos e a produção de evidências mais detalhadas do que os anteriores.

Estas questões apresentam, também, desafios para as entidades aprovadoras, que passarão a ter de se especializar cada vez mais para ganhar competências e conhecimento no grande leque tecnológico de produtos que podem ser classificados como dispositivos médicos.

Quando questionada sobre as especificidades de um processo de aprovação, a entrevistada referiu que era complicado determinar uma vez que cada um apresenta as suas particularidades, nomeadamente no que diz respeito ao nível de segurança e finalidade prevista. Ainda assim, sublinhou a importância do sistema de gestão de qualidade e de risco postos em prática pela organização candidata. Complementarmente, destacou a necessidade da existência de avaliações clínicas que fundamentem a finalidade do dispositivo.

A conversa avançou para a realidade atual do *software* no mundo dos dispositivos médicos. A Mariana Madureira aprontou-se em admitir esta realidade. De acordo com a mesma, existem já mais de 600 dispositivos médicos de pleno direito constituídos apenas por *software* registados em Portugal, sendo que «o número é crescente de ano para ano.». A entrevistada considerou que esta é uma evolução natural de acordo com as mais recentes tendências tecnológicas e o novo regulamento veio a providenciar meios para promover estes desenvolvimentos. *Big Data*, *Internet of Things* e *mHealth* são exemplos de «chavões» muito discutidos a nível europeu e que já são reconhecidos no âmbito legal dos dispositivos médicos. Por outro lado, a entrevistada frisou a importância de haver uma estrutura de controlo que consiga lidar com «todo um mundo de nova complexidade que se avizinha.».

A temática final da entrevista incidiu sobre a possibilidade de prescrição de *software* de um médico para um paciente como dispositivo médico, assim como uma eventual comparticipação do mesmo por parte do Estado. A interrogada referiu que «essa é uma realidade muito falada a nível europeu» no contexto dos dispositivos médicos. A mesma admitiu que é uma possibilidade muito real, apesar de não conseguir dar certezas se já existe algum exemplo que se verifique.

Terminou referindo que essa possibilidade implicaria protocolos com outras entidades, como é o caso dos Serviços Partilhados do Ministério da Saúde [116], proprietários da plataforma digital responsável por gerir a prescrição dos dispositivos. É necessário também ultrapassar uma certa insegurança por parte dos

utilizadores com a utilização deste tipo de aplicações.

7.3 Discussão dos Resultados

Relativamente à primeira abordagem, de implementação do ciclo de vida proposto em ambiente académico, os resultados não foram conclusivos principalmente devido à incapacidade de implementar um ciclo de vida verdadeiramente ágil sem que se reunissem as condições necessárias para ter uma equipa inteiramente dedicada ao projeto. Os elementos da equipa eram estudantes que, naturalmente, tinham outras prioridades como outras unidades curriculares e até mesmo outras atividades profissionais. Sendo assim, uma das principais conclusões a retirar é a de que qualquer ciclo de vida de desenvolvimento de *software*, mas em particular os ciclos de vida ágeis, apenas funcionam quando existe um grau de compromisso elevado com o processo. Especialmente numa tentativa de validação, é importante que todas as etapas sejam seguidas, para que os resultados possam ser conclusivos e confiáveis.

Ainda assim, é possível retirar algumas considerações importantes:

- A equipa não teve dificuldade em produzir documentação necessária para a produção de evidências, nem em atualizar esta documentação quando necessário, uma vez que compreendeu que a mesma trouxe valor para o cliente e para o projeto.

- A equipa utilizou, de forma autónoma, ferramentas de gestão de configurações como uma plataforma de *issue tracking*, um sistema de versionamento de código e uma ferramenta de integração contínua, mesmo que estas estratégias não tivessem sido especificamente direcionadas para finalidades de gestão de configurações, mas sim para facilitar o trabalho da equipa.

A segunda abordagem permitiu direcionar a validação, por um lado, mais para um setor profissional da indústria e, por outro, para os tópicos mais específicos abordados nesta dissertação como o próprio ciclo de vida que foi apresentado. Da série de conversas conduzidas junto de profissionais da área da engenharia e desenvolvimento de sistemas críticos, destacam-se os seguintes apontamentos:

- Foram referidas, como principais barreiras para o desenvolvimento *agile* no contexto de sistemas críticos, os seguintes tópicos: características do cliente, dependências de sistemas externos, dificuldade em produzir evidências aceitáveis para processos de auditoria (nomeadamente documentação), rastreabilidade e dificuldade em implementar um processo de gestão de modificações eficiente e compatível com os requisitos das normas e regulamentos.
  Estes tópicos vão de encontro àquilo que foi pesquisado na bibliografia [52, 70, 74]. As questões relacionadas com a natureza do cliente são, de facto, difíceis de ultrapassar e o ciclo de vida de

desenvolvimento tem de ser escolhido de forma a se ajustar com a mesma. Porém, com a introdução do RDM, é expectável que sejam desenvolvidos mais sistemas cuja audiência é mais ampla do que um cliente em particular e este é o contexto onde se pretende inserir este trabalho de pesquisa. Adicionalmente, é importante acrescentar que um ciclo iterativo promove a naturalidade de implementação de modificações, atenuando esta dificuldade expressa pelo Vítor Vieira. Para endereçar os restantes tópicos, apresentam-se estratégias no capítulo 6.

- Foi possível concluir que todos os entrevistados reconhecem o valor das metodologias ágeis para a produtividade dos projetos de *software* e a natureza iterativa das mesmas foi considerada, por todos, como um dos maiores benefícios do *agile*.

- As estratégias de *tailoring* foram classificadas como importantes por todos os participantes nas entrevistas. Ainda assim, dois dos entrevistados referiram que estes processos deverão ser feitos com responsabilidade, sem desvirtuar os modelos originais.

- Os quatro entrevistados sublinharam a importância das práticas de *quality assurance*. Dois dos mesmos (aqueles com experiência de desenvolvimento) referiram que estas práticas ajudam na implementação de processo que seriam negligenciados de outra forma, enquanto que os dois restantes (com mais experiência de gestão) falaram de evitar risco e introduzir confiança na solução.

- A conjugação dos três itens anteriores permite validar a metodologia apresentada, que consiste na apresentação de um ciclo de vida ágil com introdução de itens de *tailoring*. Validam-se também, os principais objetivos deste modelo, que são de agilizar o desenvolvimento de sistemas no domínio dos dispositivos médicos, assegurando práticas de *quality assurance* suficientes para garantir *compliance* com um conjunto de normas específicas.

- A conversa com a Mónica Sobreira relativamente à gestão do risco foi de encontro à abordagem apresentada nesta dissertação, na medida em que a entrevistada enfatizou uma abordagem de *continuous improvement* que permitisse encarar o risco com naturalidade no decorrer do processo. Considera-se que a análise do risco no decorrer das cerimónias do *SCRUM* de uma forma iterativa e contínua se ajusta a esta definição.

- No que diz respeito às ferramentas de auxílio à produção de evidências, aquilo que foi discutido com os intervenientes também se adequou com o conteúdo apresentado no capítulo 6. Todos os interrogados referiram uma plataforma de *issue tracking* como fundamental para a produção de evidências e um dos entrevistados referiu o sistema de controlo de versões de código.
  Adicionalmente, três dos quatro participantes reforçaram a necessidade de automatizar atividades mais rotineiras no processo de desenvolvimento, ideia esta que vai de encontro à tese apresentada.

- Finalmente, no que diz respeito à apresentação do modelo de ciclo de vida proposto, verificou-se que nenhum dos participantes revelou objeções impeditivas de uma implementação bem sucedida do mesmo. Enquanto que a Mónica Sobreira considerou que poderia haver ainda mais evidências geradas automaticamente, o Patrick Machado considerou que talvez fossem necessárias mais evidências formais, nomeadamente de testes. É importante ainda ressalvar que o Vítor Vieira deu a sugestão para uma atividade adicional no ciclo, a que apelidou de *market research*.

O passo final na etapa de validação foi a conversa com um elemento da autoridade competente nacional, o INFARMED, da qual surgiram os seguintes tópicos de discussão:

- O RDM foi construído para endereçar os avanços tecnológicos emergentes na área dos dispositivos médicos, que alteraram a natureza dos mesmos e geraram a necessidade de uma adaptação quer por parte dos fabricantes, quer por parte das entidades avaliadoras.
  A Comissão Europeia, mediante as mudanças no ecossistema tecnológico nesta indústria, sentiu a necessidade de introduzir novos regulamentos que permitissem assegurar a compreensão de todos em relação à terminologia relevante, e também garantir segurança.

- O processo de aprovação dos dispositivos médicos é bastante complexo e exigente, realizado por profissionais cada vez mais especializados de acordo com requisitos cada vez mais rigorosos. Isto justifica a necessidade de um ciclo de vida de desenvolvimento bem definido e desenhado para responder a todos os critérios das normas em questão.

- O desenvolvimento de sistemas *standalone* de *software* é uma realidade a nível nacional e está a presenciar um crescimento vertiginoso. Esta confirmação foi bastante importante para validar a motivação desta dissertação. Conclui-se que, de facto, existe uma mudança no ecossistema tecnológico dos dispositivos médicos, na medida em que existem produtos de *software* que podem ser construídos e certificados como tal. A entrevistada falou, inclusivamente, em conceitos como aplicações *web* e móveis que vão precisamente de encontro ao tipo de aplicações identificadas nesta dissertação como um ponto de crescimento nesta área.

- Não existe nada que impeça, em Portugal, a prescrição e comparticipação de aplicações de *software* classificadas como dispositivos médicos de pleno direito. Este assunto é presentemente motivo de grande discussão a nível europeu. Esta questão pode ser importantíssima para os fabricantes de *software* na indústria médica. É bastante entusiasmante a ideia de que é possível tornar os cuidados de saúde cada vez mais acessíveis e eficazes através do *software*.

8

Conclusão e Trabalho Futuro

A realidade dos dispositivos médicos no panorama europeu encontra-se, atualmente, numa vertiginosa mudança rumo à era digital. Com a introdução do novo Regulamento para os Dispositivos Médicos 2017/745/EEC por parte do Parlamento Europeu e do Conselho, esta mudança ganhou forma e reuniram-se todas as condições para que os fabricantes de dispositivos médicos possam desenvolver produtos exclusivamente *software* neste contexto, com finalidades de prognóstico e diagnóstico.

Esta dissertação concentrou-se no estudo das implicações desta mudança, nomeadamente no que diz respeito aos processos, ciclos de vida e ferramentas de desenvolvimento de *software* aplicados especificamente aos dispositivos médicos.

Suplementarmente, foi proposto, não só um modelo de ciclo de vida desenhado para endereçar o problema baseado na metodologia SCRUM, mas também estratégias específicas de desenvolvimento direcionadas ao cumprimento da normas internacionais aplicáveis.

8.1 Conclusões

O novo Regulamento para os Dispositivos Médicos foi introduzido em substituição das diretivas da «nova abordagem» por um motivo bastante claro: a terminologia utilizada nas antigas diretivas em vigor já se encontrava desatualizada relativamente aos produtos e técnicas em utilização para o fabrico de dispositivos médicos, devido aos avanços tecnológicos que tornaram o *software* cada vez mais preponderante e relevante neste âmbito.

Como consequência do reconhecimento dos sistemas isolados de *software* como dispositivos médicos de pleno direito, assim como da introdução das finalidades previstas de diagnóstico e previsão, reúnem-se as condições para desenvolver soluções com características mais *lightweight* e âmbitos mais diversificados e flexíveis (nomeadamente no que diz respeito aos *stakeholders* envolvidos). Confirmou-se, inclusivamente, que é real a possibilidade de prescrição de uma aplicação de *software* de médico para paciente em contextos específicos. Adicionalmente, a comparticipação deste sistema pode acontecer da mesma forma que aconteceria para qualquer outro dispositivo médico.

Assim como houve necessidade de atualização numa componente legal, há também esta necessidade para os processos de desenvolvimento de *software* na indústria dos sistemas críticos, especificamente no setor dos dispositivos médicos. É urgente tirar partido do reconhecimento dado e utilizar modelos de desenvolvimento que estimulem a produtividade e resistam às características instáveis e de mudança constante do presente ecossistema tecnológico.

A norma IEC 62304:2006 é, no momento desta dissertação, a indicada para fornecer as recomendações necessárias para os ciclos de vida de desenvolvimento de *software* aplicado aos dispositivos médicos. Esta norma não inclui qualquer tipo de restrição relativamente à natureza do modelo de ciclo de vida a utilizar.

As metodologias de desenvolvimento ágil, como o *SCRUM*, têm características benéficas em ambientes de constante incerteza. O *agile* acelera a entrega de valor de negócio através de um processo de planeamento e *feedback* contínuo, reduzindo o risco associado à mudança.

Não existe qualquer motivo objetivo que impeça ou mesmo desmotive a utilização de ciclos de vida ágeis no contexto dos sistemas críticos, caso estejam reunidas todas as condições para tal (como é o caso das características do cliente). Ainda assim, é importante que o modelo escolhido consiga satisfazer todos os requisitos impostos pelo Regulamento para os Dispositivos Médicos, bem como pela norma IEC 62304:2006, requisitos estes desenhados para garantir a segurança dos dispositivos. Para tal, é fundamental a utilização de uma estratégia de *tailoring* que permita ultrapassar as principais barreiras identificadas à utilização dos métodos ágeis como a produção de documentação, rastreabilidade e *quality assurance*.

Foi proposto um modelo para o ciclo de vida baseado no *SCRUM*, relativamente ao qual foram introduzidos itens de *tailoring* especificamente para atingir o cumprimento dos requisitos da norma IEC

62304:2006. Há confiança que este modelo estimularia uma produtividade do desenvolvimento de soluções de *software* superior às metodologias tradicionais utilizados nesta indústria, sem introduzir *overhead* adicional assinalável.

8.2 Contribuições

Foi feito um estudo aprofundado do impacto do RDM (2017/745/EEC), assim como as mudanças na legislação dos dispositivos médicos ao longo do tempo. Além disso, foi feita uma pesquisa direcionada para a realidade portuguesa no que diz respeito às implicações do mesmo regulamento.

Adicionalmente, foi estudado o ecossistema de normas internacionais influentes no desenvolvimento de *software* orientado aos dispositivos médicos, com foco no *standard* 62304:2006 que contém as recomendações específicas para os processos do ciclo de vida de desenvolvimento neste mesmo âmbito.

É defendida a tese de que a evolução natural dos processos de desenvolvimento de *software* para a indústria dos dispositivos médicos são as metodologias ágeis, tendo em conta a direção tecnológica que é expressa pelo novo enquadramento legal europeu. Neste sentido, é também apresentado, nesta dissertação, um novo ciclo de vida de desenvolvimento de *software*, baseado no *tailoring* do modelo *SCRUM* para melhor se ajustar às exigências da norma supramencionada e fornecer novas potencialidades para a construção de *software* tendo em conta a nova realidade dos dispositivos médicos.

Relativamente ao novo modelo de ciclo de vida apresentado, são contribuições tangíveis:

1. Um relatório de auditoria interna direcionado ao cumprimento da norma IEC 62304:2006 e ao modelo *agile* apresentado (Anexo B);

2. Uma *fact-sheet* para fácil entendimento e implementação do modelo *agile* proposto (Anexo C);

O trabalho produzido no âmbito desta dissertação resultou, também, na seguinte publicação:

- M. Zamith and G. Gonçalves, «Towards an Agile development model for certifiable medical device software: Taking advantage of the Medical Device Regulation» 13th International Joint Conference on Software Technologies (ICSOFT), Porto, 2018.

8.3 Trabalho Futuro

Uma das formas de aprofundar o conhecimento adquirido no tema seria o estudo adicional, de um ponto de vista legal, do Regulamento para os Dispositivos Médicos introduzido em 2017. É importante conhecer os detalhes do processo de aprovação orientado pelas autoridades competentes, assim como pelos

organismos notificados, de forma a compreender o grau de exigência do mesmo e a natureza concreta das evidências que estas procuram no processo dos fabricantes.

Em relação ao modelo de ciclo de vida proposto, é importante considerar as sugestões que foram feitas pelos intervenientes nas entrevistas apresentadas no capítulo 7.2 de forma a melhorar o trabalho desenvolvido e até procurar mais contribuições, nomeadamente de pessoas com experiência profissional mais específica ao problema em debate.

Especificamente, foram dadas as seguintes sugestões para melhoria do modelo apresentado:

- Introdução de uma atividade apelidada de «pesquisa de mercado», que responderia a uma exigência do MDR para identificar anomalias em produtos semelhantes e gerar medidas de correção no caso das mesmas serem aplicáveis ao projeto;
- Inclusão de uma tarefa de análise de risco também na cerimónia de *sprint retrospective*;
- Estudar a possibilidade de eliminar por completo a produção de documentação no interior da *sprint* de forma manual, automatizando a produção ou atualização dos documentos de ERS e CIL;

Complementarmente, seria essencial validar a metodologia proposta num contexto de desenvolvimento real de um dispositivo médico de Classe de segurança B ou C. Este processo de teste permitiria tirar conclusões com características mais objetivas e quantitativas, por exemplo, através da comparação com métricas existentes em estudos encontrados na bibliografia. Por exemplo, Abrahamsson *et al.* [117] apresentaram um estudo comparativo no qual ciclos de vida de desenvolvimento de *software* são analisados e avaliados. Além disso, Boehm [64] apresentou uma comparação detalhada entre diversos modelos diferentes de forma a estudar o equilíbrio entre «agilidade» e disciplina dos mesmos. o autor recolheu métricas como desempenho, cumprimento da agenda proposta, satisfação da equipa e taxa de defeitos encontrados.

Bibliografia

[1] John C. Knight. Safety Critical Systems: Challenges and Directions. In *Proceedings of the 24th international conference on Software engineering - ICSE '02*, page 547, New York, New York, USA, 2002. ACM Press. URL: http://portal.acm.org/citation.cfm?doid=581339.581406, doi:10.1145/581339.581406.

[2] Jonathan Bowen and Victoria Stavridou. Safety-critical systems, formal methods and standards. *Software Engineering Journal*, 8(4):189, 1993. URL: http://digital-library.theiet.org/content/journals/10.1049/sej.1993.0025, doi:10.1049/sej.1993.0025.

[3] NASA. Mars Climate Orbiter Team Finds Likely Cause Of Loss, 1999. URL: https://mars.nasa.gov/msp98/news/mco990930.html.

[4] Dolores R. Wallace and D. Richard Kuhn. Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data. *International Journal of Reliability, Quality and Safety Engineering*, 08(04):351–371, dec 2001. URL: http://www.worldscientific.com/doi/abs/10.1142/S021853930100058X, doi:10.1142/S021853930100058X.

[5] N.G. Leveson and C.S. Turner. An investigation of the Therac-25 accidents. *Computer*, 26(7):18–41, jul 1993. URL: http://ieeexplore.ieee.org/document/274940/, doi:10.1109/MC.1993.274940.

[6] Stephen Allen. Medical device software under the microscope. *Network Security*, 2014(2):11–12, feb 2014. URL: http://linkinghub.elsevier.com/retrieve/pii/S1353485814700212, doi:10.1016/S1353-4858(14)70021-2.

[7] Jeff C. Goldsmith. The healthcare information technology sector. In Lawton Robert Burns, editor, *The Business of Healthcare Innovation*, pages 322–347. Cambridge University Press, Cambridge, 2006. URL: http://ebooks.cambridge.org/ref/id/CBO9781139176620A012, doi:10.1017/CBO9780511488672.008.

[8] Boris Gloger. 3 Reasons for Introducing Agile Product Development in Medical Technology. Technical report. URL: https://borisgloger.com/wp-content/uploads/2016/09/Whitepaper{_}MedTec{_}en.pdf.

[9] Kurt H. Kruger and Max A. Kruger. The medical device sector. In Lawton Robert Burns, editor, *The Business of Healthcare Innovation*, pages 376–450. Cambridge University Press, Cam-

bridge, 2012. URL: http://ebooks.cambridge.org/ref/id/CBO9781139176620A013, doi:10.1017/CBO9781139176620.006.

[10] Peter F. Drucker. *Managing in the Next Society*. New York, New York, USA, 2003.

[11] M. Zema, S. Rosati, V. Gioia, M. Knaflitz, and G. Balestra. Developing medical device software in compliance with regulations. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, volume 2015-Novem, pages 1331–1334. IEEE, aug 2015. URL: http://ieeexplore.ieee.org/document/7318614/, doi:10.1109/EMBC.2015.7318614.

[12] David Vogel. *Medical Device Software: Verification, Validation and Compliance*. Artech House, 2011.

[13] Paul Quinn. The EU commission's risky choice for a non-risk based strategy on assessment of medical devices. *Computer Law & Security Review*, 33(3):361–370, jun 2017. URL: http://linkinghub.elsevier.com/retrieve/pii/S0267364916301637, doi:10.1016/j.clsr.2017.03.019.

[14] Martin McHugh, Fergal McCaffery, and Garret Coady. An Agile Implementation within a Medical Device Software Organisation. In *Software Process Improvement and Capability Determination*, pages 190–201. 2014. URL: http://link.springer.com/10.1007/978-3-319-13036-1{_}17, doi:10.1007/978-3-319-13036-1_17.

[15] Diretiva 2007/47/EC do Parlamento Europeu e do Conselho que altera a diretiva do conselho 90/385/EEC relativa à aproximação das legislações dos Estados-membros respeitantes aos dispositivos medicinais implantáveis ativos, a diretiva do conselho 93/42/EEC relativa a dispositivos médicos e a diretiva 98/8/EC relativa a colocação de produtos biocidas no mercado. Diretiva, União Europeia, September 2007. URL: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2007:247:0021:0055:en:PDF.

[16] Regulamento (UE) 2017/745 do Parlamento Europeu e do Conselho relativo aos dispositivos médicos, que altera a Diretiva 2001/83/CE, o Regulamento (CE) n.o 178/2002 e o Regulamento (CE) n.o 1223/2009 e que revoga as Diretivas 90/385/CEE e 93/42/CEE do Conselho. Regulamento, União Europeia, April 2017. URL: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32017R0745.

[17] IEC 62304 :2006. Medical device software – Software life cycle processes. Standard, IEC, May 2006. URL: https://www.iso.org/standard/38421.html.

[18] ISO/IEC 12207:1995. Information technology – Software life cycle processes. Standard, ISO, IEC, July 1995. URL: https://www.iso.org/standard/21208.html.

[19] Corinna Sorenson and MIichael Drummond. Improving Medical Device Regulation: The United States and Europe in Perspective. *Milbank Quarterly*, 92(1):114–150, mar 2014. URL: http://doi.wiley.com/10.1111/1468-0009.12043, doi:10.1111/1468-0009.12043.

[20] Diretiva 90/385/EEC do Conselho relativa à aproximação das legislações dos Estados-membros respeitantes aos dispositivos medicinais implantáveis activos. Diretiva, União Europeia, October 2007. URL: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CONSLEG:1990L0385:20071011:en:PDF.

[21] Diretiva 93/42/EEC do Conselho relativa aos dispositivos médicos. Diretiva, União Europeia, October 2007. URL: http://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1512833106092&uri=CELEX:31998L0079.

[22] Directiva 98/79/CE do Parlamento Europeu e do Conselho relativa aos dispositivos médicos de diagnóstico in vitro. Diretiva, União Europeia, October 2007. URL: http://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1512833106092&uri=CELEX:31998L0079.

[23] Infarmed. Comissão de avaliação dispositivos médicos - INFARMED, I.P. URL: http://www.infarmed.pt/web/infarmed/institucional/estrutura-e-organizacao/comissoes-tecnicas-especializadas/comissao-de-avaliacao-dispositivos-medicos-diagnostico-vitro.

[24] Daniel B Kramer, Shuai Xu, and Aaron S Kesselheim. Regulation of Medical Devices in the United States and European Union. *New England Journal of Medicine*, 366(9):848–855, mar 2012. URL: http://www.nejm.org/doi/abs/10.1056/NEJMhle1113918, doi:10.1056/NEJMhle1113918.

[25] Minna Pikkarainen. Agile Medical Device Software Development: Introducing Agile Practices into MDevSPICE®. *International Journal On Advances in Life Sciences*, 8(1):133–142, 2016.

[26] Fabio Geremia. Quality aspects for medical devices, quality system and certification process. *Microchemical Journal*, 136:300–306, jan 2018. URL: http://linkinghub.elsevier.com/retrieve/pii/S0026265X17303521, doi:10.1016/j.microc.2017.04.018.

[27] ISO 13485:2016. Medical devices – Quality management systems – Requirements for regulatory purposes. Standard, ISO, March 2016. URL: https://www.iso.org/standard/59752.html.

[28] ISO 9001:2015. Quality management systems – Requirements. Standard, ISO, May 2015. URL: https://www.iso.org/standard/62085.html.

[29] ISO/IEC 90003:2014. Software engineering – Guidelines for the application of ISO 9001:2008 to computer software. Standard, ISO, IEC, December 2014. URL: https://www.iso.org/standard/66240.html.

[30] ISO/IEC 15504-5:2012. Information technology – Process assessment – Part 5: An exemplar software life cycle process assessment model. Standard, ISO, February 2012. URL: https://www.iso.org/standard/60555.html.

[31] International Medical Device Regulators Forum. Statement regarding Use of IEC 62304:2006 "Medical device software - Software life cycle processes". Technical report, 2015. URL: http://www.imdrf.org/docs/imdrf/final/procedural/imdrf-proc-151002-medical-device-software-n35.pdf.

[32] ISO 14971:2007. Medical devices – Application of risk management to medical devices. Standard, ISO, March 2007. URL: https://www.iso.org/standard/38193.html.

[33] IEC/TR 80002-1:2009. Medical device software – Part 1: Guidance on the application of ISO 14971 to medical device software. Standard, IEC, September 2009. URL: https://www.iso.org/standard/54146.html.

[34] IEC 60601-1-11:2015. Medical electrical equipment – Part 1-11: General requirements for basic safety and essential performance – Collateral standard: Requirements for medical electrical equipment and medical electrical systems used in the home healthcare environment. Standard, ISO, January 2015. URL: https://www.iso.org/standard/65529.html.

[35] Martin McHugh, Fergal McCaffery, and Valentine Casey. Standalone Software as an Active Medical Device. In *Communications in Computer and Information Science*, volume 155 CCIS, pages 97–107. 2011. URL: http://link.springer.com/10.1007/978-3-642-21233-8{_}9, doi:10.1007/978-3-642-21233-8_9.

[36] IEC 82304-1:2016. Health software – Part 1: General requirements for product safety. Standard, ISO, January 2016. URL: https://www.iso.org/standard/59543.html.

[37] FibriCheck. FibriCheck - Physicians. URL: https://fibricheck.com/physicians.

[38] Michael Lang. Heart Rate Monitoring Apps: Information for Engineers and Researchers About the New European Medical Devices Regulation 2017/745. *JMIR Biomedical Engineering*, 2(1):e2, aug 2017. URL: http://biomedeng.jmir.org/2017/1/e2/, doi:10.2196/biomedeng.8179.

[39] Choong Ho Lee and Hyung-Jin Yoon. Medical big data: promise and challenges. *Kidney Research and Clinical Practice*, 36(1):3–11, mar 2017. URL: http://www.krcp-ksn.org/journal/view.html?doi=10.23876/j.krcp.2017.36.1.3, doi:10.23876/j.krcp.2017.36.1.3.

[40] Infarmed. Dispositivos médicos - INFARMED, I.P. URL: http://www.infarmed.pt/web/infarmed/entidades/dispositivos-medicos.

[41] Infarmed. Avaliação da Conformidade - INFARMED, I.P. URL: http://www.infarmed.pt/web/infarmed/entidades/dispositivos-medicos/avaliacao-da-conformidade.

[42] Decreto-Lei n.º 145/2009 de 17 de Junho, 2009.

[43] Decreto-Lei n.º 97/2015 , de 1 de junho, 2015.

[44] Roger S Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 7th edition, 2009. arXiv:arXiv:1011.1669v3, doi:10.1017/CBO9781107415324.004.

[45] IEEE. EEE Standards Collection: Software Engineering. In *IEEE Standard 610.12-1990*. 1993.

[46] Ian Sommerville. *Software Engineering*. Lecture Notes in Computer Science. Pearson, Berlin, Heidelberg, 2010. URL: http://link.springer.com/10.1007/3-540-57209-0, doi:10.1007/3-540-57209-0.

[47] Pierre Bourque and Richard E. Fairley. *Guide to the Software Engineering - Body of Knowledge*. IEEE Computer Society Press, 2014. URL: www.swebok.org, arXiv:arXiv:1210.1833v2, doi:10.3403/30314312u.

[48] Dr. Winston W. Royce. Managing the Development of large Software Systems. *Ieee Wescon*, (August):1–9, 1970.

[49] Thomas E. Bell and T. A. Thayer. Software Requirements: Are They Really a Problem? *ICSE*, 1976.

[50] Kevin Forsberg and Harold Mooz. The Relationship of System Engineering to the Project Cycle. *INCOSE International Symposium*, 1(1):57–65, oct 1991. URL: http://doi.wiley.com/10.1002/j.2334-5837.1991.tb01484.x, doi:10.1002/j.2334-5837.1991.tb01484.x.

[51] Paul Rook. Controlling software projects. *Software Engineering Journal*, 1(1):7–16, 1986. doi:10.1049/sej:19860003.

[52] Fergal McCaffery, Kitija Trektere, and Ozden Ozcan-Top. Agile – Is it Suitable for Medical Device Software Development? In *Software Process Improvement and Capability Determination*, volume 770 of *Communications in Computer and Information Science*, pages 417–422. Springer International Publishing, 2016. URL: http://link.springer.com/10.1007/978-3-319-38980-6{_}30, doi:10.1007/978-3-319-38980-6_30.

[53] C. Larman and V.R. Basili. Iterative and incremental developments. A brief history. *Computer*, 36(6):47–56, jun 2003. URL: http://ieeexplore.ieee.org/document/1204375/, doi:10.1109/MC.2003.1204375.

[54] A.M. Davis. Operational prototyping: a new development approach. *IEEE Software*, 9(5):70–78, sep 1992. URL: http://ieeexplore.ieee.org/document/156899/, doi:10.1109/52.156899.

[55] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, may 1988. URL: http://ieeexplore.ieee.org/document/59/, doi:10.1109/2.59.

[56] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile Software Development, 2001. URL: http://agilemanifesto.org/.

[57] Shane Warden and James Shore. *The Art of Agile Development*. O'Reilly Media, 2008.

[58] Alistair Cockburn. *Agile Software Development*, volume 5. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=947100http://link.springer.com/10.1007/978-3-642-12575-1, doi:10.1007/978-3-642-12575-1.

[59] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co, 2009.

[60] Ken Schwaber. SCRUM Development Process. In *Business Object Design and Implementation*, volume 6, pages 117–134. Springer London, London, oct 1997. URL: http://link.springer.com/10.1007/978-1-4471-0947-1{_}11, doi:10.1007/978-1-4471-0947-1_11.

[61] Jeff Sutherland and JJ Sutherland. *Scrum: The Art of Doing Twice the Work in Half the Time*. Currency, 2014.

[62] Ken Schwaber and Jeff Sutherland. The Scrum Guide. Technical report, 2017.

[63] Manifesto for Software Craftsmanship. URL: http://manifesto.softwarecraftsmanship.org/.

[64] Barry Boehm. Balancing Agility and Discipline: A Guide for the Perplexed. pages 1–1. 2004. URL: http://link.springer.com/10.1007/978-3-540-24675-6{_}1, arXiv:arXiv:1011.1669v3, doi:10.1007/978-3-540-24675-6_1.

[65] Maged N. Kamel Boulos, Ann C. Brewer, Chante Karimkhani, David B. Buller, and Robert P. Dellavalle. Mobile medical and health apps: state of the art, concerns, regulatory control and certification. *Online Journal of Public Health Informatics*, 5(3):1–23, feb 2014. URL: http://journals.uic.edu/ojs/index.php/ojphi/article/view/4814, arXiv:ISSN-1529-1944, doi:10.5210/ojphi.v5i3.4814.

[66] E Ray Dorsey, Yu-feng Yvonne Chan, Michael V Mcconnell, Stanley Y Shaw, Andrew D Trister, and Stephen H Friend. The Use of Smartphones for Health Research. *Academic Medicine*, 92(2):157–160, 2017. doi:10.1097/ACM.0000000000001205.

[67] Darrell K. Rigby, Jeff Sutherland, and Hirotaka Takeuchi. Embracing Agile. URL: https://hbr.org/2016/05/embracing-agile.

[68] TechBeacon. Agile vs. waterfall: Survey shows agile is now the norm. URL: https://techbeacon.com/survey-agile-new-norm.

[69] D Vogel. Agile Methods: Most are not ready for prime time in medical device software design and development. *DesignFax Online*, (July):1–5, 2006. URL: http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Agile+Methods:+Most+are+not+ready+for+prime+time+in+medical+device+software+design+and+development{#}0.

[70] Martin McHugh, Fergal McCaffery, and Valentine Casey. Barriers to Adopting Agile Practices When Developing Medical Device Software. In *Software Process Improvement and Capability Determination*, pages 141–147. 2012. URL: http://link.springer.com/10.1007/978-3-642-30439-2{_}13, doi:10.1007/978-3-642-30439-2_13.

[71] Martin McHugh, Fergal McCaffery, Brian Fitzgerald, Klaas-Jan Stol, Valentine Casey, and Garret Coady. Balancing Agility and Discipline in a Medical Device Software Organisation. In *Software Process Improvement and Capability Determination*, number June, pages 199–210. 2013. URL: http://link.springer.com/10.1007/978-3-642-38833-0{_}18, doi:10.1007/978-3-642-38833-0_18.

[72] Christian Denger, Raimund L Feldmann, Martin Host, Christin Lindholm, and Forrest Shull. A Snapshot of the State of Practice in Software Development for Medical Devices. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 485–487. IEEE, sep 2007. URL: http://ieeexplore.ieee.org/document/4343787/, doi:10.1109/ESEM.2007.54.

[73] S.B. Leif and R.C. Leif. Producing quality software according to medical regulations for devices. In *[1992] Proceedings Fifth Annual IEEE Symposium on Computer-Based Medical Systems*, volume 510, pages 265–272. IEEE Comput. Soc. Press, 1992. URL: http://ieeexplore.ieee.org/document/244939/, doi:10.1109/CBMS.1992.244939.

[74] Martin McHugh and Fergal McCaffery. Challenges experienced by Medical Device Software Development Organizations while following a Plan-Driven Software Development Life Cycle. *EuroSPI 2013*, pages 25–27, 2013.

[75] Jason Bowers, John May, Erik Melander, Matthew Baarman, and Azeem Ayoob. Tailoring XP for Large System Mission Critical Software Development. *Extreme Programming and Agile Methods — XP/Agile Universe 2002 SE - 10*, 2418:100–111, 2002. URL: http://dx.doi.org/10.1007/3-540-45672-4{_}10, doi:10.1007/3-540-45672-4_10.

[76] P Manhart and K Schneider. Breaking the ice for agile development of embedded software: an industry experience report. In *Proceedings. 26th International Conference on Software Engineering*, pages 378–386. IEEE Comput. Soc, 2004. URL: http://ieeexplore.ieee.org/document/1317460/, doi:10.1109/ICSE.2004.1317460.

[77] J.W. Spence. There has to be a better way! [software development]. In *Agile Development Conference (ADC'05)*, pages 272–278. IEEE Comput. Soc, 2005. URL: http://ieeexplore.ieee.org/document/1609832/, doi:10.1109/ADC.2005.47.

[78] Pieter Adriaan Rottier and Victor Rodrigues. Agile Development in a Medical Device Company. In *Agile 2008 Conference*, pages 218–223. IEEE, 2008. URL: http://ieeexplore.ieee.org/document/4599480/, doi:10.1109/Agile.2008.52.

[79] Rod Rasmussen, Tim Hughes, J.R. Jenks, and John Skach. Adopting Agile in an FDA Regulated Environment. In *2009 Agile Conference*, pages 151–155. IEEE, aug 2009. URL: http://ieeexplore.ieee.org/document/5261092/, doi:10.1109/AGILE.2009.50.

[80] Xiaocheng Ge, Richard F Paige, and John A. McDermid. An Iterative Approach for Development of Safety-Critical Software and Safety Arguments. In *2010 Agile Conference*, pages 35–43. IEEE, aug 2010. URL: http://ieeexplore.ieee.org/document/5562808/, doi:10.1109/AGILE.2010.10.

[81] Martin McHugh, Fergal McCaffery, and Valentine Casey. Integrating Agile Practices with a Medical Device Software Development Lifecycle. *EuroSPI 2012*, 2012. URL: http://ulir.ul.ie/handle/10344/2524.

[82] Martin McHugh. Integrating Agile Practices with Plan-Driven Medical Device Software Development. In *13th International Conference on Agile Software Development: XP 2012 Doctoral Symposium*, 2012. URL: http://ulir.ul.ie/handle/10344/2524.

[83] Martin McHugh, Oisin Cawley, Fergal McCaffcry, Ita Richardson, and Xiaofeng Wang. An agile V-model for medical device software development to overcome the challenges with plan-driven software development lifecycles. In *2013 5th International Workshop on Software Engineering in Health Care (SEHC)*, pages 12–19. IEEE, may 2013. URL: http://ieeexplore.ieee.org/document/6602471/, doi:10.1109/SEHC.2013.6602471.

[84] Brian Fitzgerald, Klaas-Jan Stol, Ryan O ' Sullivan, and Donal O 'brien. Scaling Agile Methods to Regulated Environments: An Industry Case Study. *Proceedings of the 2013 International Conference on Software Engineering*, pages 863–872, 2013. URL: https://ulir.ul.ie/bitstream/handle/10344/3055/Fitzgerald{_}2013{_}agile.pdf?sequence=2.

[85] K. Trektere, F. McCaffery, and M. Lepmets. Development of a software process framework to assist organizations developing mobile medical apps. In *2016 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 461–464. IEEE, feb 2016. URL: http://ieeexplore.ieee.org/document/7455934/, doi:10.1109/BHI.2016.7455934.

[86] Oisín Cawley, Xiaofeng Wang, and Ita Richardson. Lean/Agile Software Development Methodologies in Regulated Environments – State of the Art. In *Lecture Notes in Business Information Processing*, volume 65 LNBIP, pages 31–36. 2010. URL: http://link.springer.com/10.1007/978-3-642-16416-3{_}4, doi:10.1007/978-3-642-16416-3_4.

[87] Andrzej Beniamin Bujok, Silvana Togneri MacMahon, Peadar Grant, Dick Whelan, William J. Rickard, and Fergal McCaffery. Approach to the development of a Unified Framework for Safety Critical Software Development. *Computer Standards & Interfaces*, 54:152–161, nov 2017. URL: http://linkinghub.elsevier.com/retrieve/pii/S0920548916301921, doi:10.1016/j.csi.2016.11.013.

[88] S.R. Rakitin. Coping with Defective Software in Medical Devices. *Computer*, 39(4):40–45, apr 2006. URL: http://ieeexplore.ieee.org/document/1620994/, doi:10.1109/MC.2006.123.

[89] Valentine Casey and Fergal Mc Caffery. A lightweight traceability assessment method for medical device software. *Journal of Software: Evolution and Process*, 25(4):363–372, apr 2013. URL: http://doi.wiley.com/10.1002/smr.571, doi:10.1002/smr.571.

[90] Valentine Casey and Fergal Mc Caffery. Med-Trace: Traceability Assessment Method for Medical Device Software Development. *Igarss 2014*, (1):1–5, 2014. arXiv:arXiv:1011.1669v3, doi:10.1007/s13398-014-0173-7.2.

[91] Fergal McCaffery, Valentine Casey, M. S. Sivakumar, Gerry Coleman, Peter Donnelly, and John Burton. Medical Device Software Traceability. In *Software and Systems Traceability*, volume 9781447122, pages 321–339. Springer London, London, 2012. URL: http://link.springer.com/10.1007/978-1-4471-2239-5{_}15, arXiv:arXiv:1011.1669v3, doi:10.1007/978-1-4471-2239-5_15.

[92] Gilbert Regan, Fergal Mc Caffery, Kevin Mc Daid, and Derek Flood. Medical device standards' requirements for traceability during the software development lifecycle and implementation of a traceability assessment model. *Computer Standards & Interfaces*, 36(1):3–9, nov 2013. URL: http://linkinghub.elsevier.com/retrieve/pii/S0920548913000780, doi:10.1016/j.csi.2013.07.012.

[93] P. Xu and B. Ramesh. Using process tailoring to manage software development challenges. *IT Professional*, 10(4):39–45, July 2008. doi:10.1109/MITP.2008.81.

[94] Sparx Systems Pty Ltd. Enterprise Architect - UML Design Tools and UML CASE tools for software development, 2018. URL: http://sparxsystems.com/products/ea/.

[95] Agile Alliance. What is Agile Software Development? URL: https://www.agilealliance.org/.

[96] Steven Fraser, Kent Beck, Bil Caputo, Tim Mackinnon, James Newkirk, and Charlie Poole. Test Driven Development. *Test*, (October):459–462, 2002. URL: http://c2.com/cgi/wiki?TestDrivenDevelopment.

[97] Peter Schuh. Agile configuration management for large organizations, 2007. URL: https://www.ibm.com/developerworks/rational/library/mar07/schuh/index.html.

[98] Git. Git, 2018. URL: https://git-scm.com/.

[99] Apache. Apache Subversion. URL: https://subversion.apache.org/.

[100] Dane Bertram, Amy Voida, Saul Greenberg, and Robert Walker. Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking Software Engineering. *Proceedings of the 2010 Computer Supported Cooperative Work Conference*, pages 291–300, 2010. doi:10.1145/1718918.1718972.

[101] Atlassian. Jira | Issue & Project Tracking Software | Atlassian, 2018. URL: https://www.atlassian.com/software/jira.

[102] Mozilla. Bugzilla, 2018. URL: https://www.bugzilla.org/.

[103] Microsoft. Visual Studio IDE, Editor de Código, Team Services e Mobile Center, 2018. URL: https://www.visualstudio.com/.

[104] JetBrains. IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains. URL: https://www.jetbrains.com/idea/.

[105] Atlassian. Crucible Code Review Tool for Git, SVN, Perforce and More, 2018. URL: https://www.atlassian.com/software/crucible.

[106] Mario Moreira. Applying Configuration Management to Agile Teams, 2008. URL: https://www.cmcrossroads.com/article/applying-configuration-management-agile-teams.

[107] Jane Cleland-Huang, Andrea Zisman, and Orlena Gotel. Software and systems traceability. *Software and Systems Traceability*, 9781447122395:1–491, 2012. arXiv:arXiv:1011.1669v3, doi:10.1007/978-1-4471-2239-5.

[108] Brad Appleton, Steve Berczuk, and Robert Cowham. Lean-Agile Traceability: Strategies and Solutions, 2017. URL: https://www.cmcrossroads.com/article/lean-agile-traceability-strategies-and-solutions.

[109] Monica Luke. How early Integration testing enables agile development, 2012. URL: https://www.ibm.com/developerworks/rational/library/early-integration-testing-enables-agile-development/index.html.

[110] Selenium. Selenium - Web Browser Automation. URL: https://www.seleniumhq.org/.

[111] Pivotal Software, Inc. Pivotal Tracker, 2018. URL: https://www.pivotaltracker.com/.

[112] GitHub. GitHub, 2018. URL: https://github.com/.

[113] TRAVIS CI, GMBH. Travis CI - Test and Deploy Your Code with Confidence, 2018. URL: https://travis-ci.org/.

[114] CRITICAL Software. CRITICAL Software - Dependable Technologies for Critical Systems, 2018. URL: https://www.criticalsoftware.com/pt/homepage.

[115] Comissão Europeia. Bodies - 93/42/EEC Medical devices, 2018. URL: http://ec.europa.eu/growth/tools-databases/nando/index.cfm?fuseaction=directive.notifiedbody&dir_id=13.

[116] SERVIÇOS PARTILHADOS DO MINISTÉRIO DA SAÚDE, EPE. Serviços Partilhados do Ministério da Saúde - SPMS, 2018. URL: https://www.criticalsoftware.com/pt/homepage.

[117] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods: Review and analysis. pages 3–107, 01 2002.

A

Estratégias de *compliance* com a norma IEC 62304:2006

Tabela A.1: Correspondência entre cada alínea da norma IEC 62304:2006 para a estratégia de *compliance* prevista no ciclo de vida de desenvolvimento apresentado.

	Descrição	Estratégia
4	Requisitos Gerais	
4.3	Classificação de segurança do *software*	Incluído no Documento de Planificação. Decisão feita na *product vision*.
5	Processo de desenvolvimento de *software*	
5.1	Planeamento de *software*	
5.1.1	Plano de desenvolvimento de *software*	Documento de Planificação de *Software* construido na *product vision*.
5.1.2	Manter o plano de desenvolvimento de *software atualizado*	Da responsabilidade do *product owner* e *scrum master*.
5.1.3	Referência do plano de desenvolvimento de *software* para o desenho do sistema e desenvolvimento	Atualização deste documento deverá fazer parte da *definition of Done*.
5.1.4	Planeamento de *standards* de desenvolvimento, métodos e ferramentas	
5.1.5	Planeamento de integração de *software* e testes de integração	
5.1.6	Planeamento de verificação de *software*	Documento de Planificação de *Software* construido na *product vision*.
5.1.7	Planeamento de gestão do risco	
5.1.8	Planeamento de documentação	
5.1.9	Planeamento de gestão de configurações de *software*	
5.1.10	Suportar itens a ser controlados	
5.1.11	Controlo de itens de configuração de *software* antes de verificação	
5.2	Análise de requisitos de *software*	
5.2.1	Definir e documentar requisitos de *software* a partir de requisitos de sistema	Documento de Especificação de Requisitos a ser construído a partir das *user stories* no decorrer da *sprint*. A atualização deste documento deve fazer parte da *definition of Done*.
5.2.2	Conteúdo dos requisitos de *software*	

5.2.3	Incluir medidas de controlo de risco nos requisitos de *software*	Risco para *user story* avaliado nas reuniões de *sprint planning*.
5.2.4	Reavaliar a análise do risco de dispositivos médicos	
5.2.5	Atualizar os requisitos de sistema	Atualização do documento de requisitos deverá fazer parte da *definition of Done*.
5.2.6	Verificar os requisitos de *software*	A verificação dos requisitos é feita da mesma forma que as *code reviews*, quando o documento é atualizado no sistema de controlo de versões.
5.3	Desenho de arquitetura de *software*	
5.3.1	Transformar requisitos de *software* em arquitetura	Documento de Arquitetura de *Software* é construído com base no *product backlog* e atualizado com base nas *user stories* e consequentes requisitos.
5.3.2	Desenvolver uma arquitetura para as interfaces dos itens de *software*	Arquitetura inicial construida na *product vision* e incrementada a cada *sprint*. Atualização da documentação deverá fazer parte da *Definition of Done*.
5.3.3	Especificar requisitos funcionais e de desempenho para itens SOUP	
5.3.4	Especificar *software* e *hardware* do sistema necessários para os itens SOUP	Especificado no documento de Especificação de Requisitos ou no documento de Arquitetura de *Software*.
5.3.5	Identificar a segregação necessária para controlo de risco	
5.3.6	Verificar arquitetura de *software*	Primeira verificação feita na *Sprint 0*. Em cada *sprint*, verificação feita em *code reviews*.
5.4	Desenho detalhado de *software*	
5.4.1	Refinar a arquitetura de *software* em unidades de *software*	
5.4.2	Desenvolver desenho detalhado para cada unidade de *software*	A ser executado no decorrer dos *sprints*, conjugado com práticas de TDD e *refactoring*.
5.4.3	Desenvolver desenho detalhado para interfaces	
5.4.4	Verificar o desenho detalhado	Através de *code reviews*.

5.5	Implementação e verificação de unidades de *software*	
5.5.1	Implementar cada unidade de *software*	Executado em cada *sprint*. Utilização de um sistema de controlo de versões.
5.5.2	Estabelecer um processo de verificação para unidades de *software*	*Code reviews*.
5.5.3	Critério de aceitação para unidade de *software*	.
5.5.4	Critérios de aceitação adicionais para unidades de *software*	Cada *user story* deverá ter um critério de aceitação (*acceptance criteria*)
5.5.5	Verificação de unidades de *software*	*Code review*. Idealmente deverá integrar com a ferramenta de gestão do projeto (ex: JIRA + Crucible).
5.6	Integração de *software* e testes de integração	
5.6.1	Integrar unidades de *software*	A executar no decorrer das *sprints*.
5.6.2	Verificar integração de *software*	Apenas se aplica a integração de *software*. Verificação feita por *code review*.
5.6.3	Testar *software* integrado	Testes de integração automáticos.
5.6.4	Conteúdo de testes de integração	Avaliação dos relatórios produzidos pelos testes de integração.
5.6.5	Verificar procedimentos de testes de integração	*Code reviews* feitas aos testes de integração automáticos.
5.6.6	Executar testes de regressão	Testes automáticos de regressão.
5.6.7	Conteúdo das evidências de testes de integração	Relatórios produzidos pelos testes automáticos.
5.6.8	Utilizar o processo de resolução de problemas de *software*	Anomalias encontradas no decorrer de testes devem ser introduzidas no *backlog* do produto.
5.7	Teste de sistema de *software*	
5.7.1	Estabelecer testes para requisitos de *software*	Testes automáticos de sistema.
5.7.2	Usar o processo de resolução de problemas de *software*	Anomalias encontradas no decorrer de testes devem ser introduzidas no *backlog* do produto.

5.7.3	Retestar após mudanças	Re-execução da bateria de testes após mudanças.
5.7.4	Verificar testes de sistema de *software*	*Code reviews* aos testes, que devem ser codificados.
5.7.5	Conteúdo dos registos de testes de sistema de *software*	Relatórios da execução dos testes de sistema.
5.8	*Release* de *software*	
5.8.1	Garantir que a verificação de *software* foi concluída	*Definition of Done* deve incluir verificação.
5.8.2	Documentar anomalias residuais conhecidas	Anomalias introduzidas no *backlog* do produto e na plataforma de gestão do projeto.
5.8.3	Avaliar anomalias residuais conhecidas	Avaliação feita tanto na reunião de *sprint planning* como *sprint review*.
5.8.4	Documentar versões entregues de *software*	Todas as *releases* devem ser documentadas no sistema de controlo de versões do projeto através de *tags*.
5.8.5	Documentar como foram entregues as versões de *software*	Deverá ser criado um relatório de *release* na plataforma de gestão do projeto.
5.8.6	Garantir que atividades e tarefas foram completadas	*Definition of Done* para todas as *stories* e para a *sprint*.
5.8.7	Arquivar software	Sistema de controlo de versões
5.8.8	Garantir repetibilidade da *release* de *software*	Automatização de *deployment* e sistema de controlo de versões asseguram repetibilidade.
6	Processo de manutenção de *software*	
6.1	Estabelecer um plano	Documentado no PDS.
6.2	Análise de problemas e modificações	
6.2.1	Documentar e avaliar *feedback*	
6.2.1.1	Monitorizar *feedback*	Da responsabilidade do *Product Owner*. Documentar *feedback* na plataforma de gestão do projeto e gerir *backlog* do produto em concordância.
6.2.1.2	Documentar e avaliar *feedback*	
6.2.1.3	Avaliar o efeito do relatório de problemas na segurança	Efetuado nas reuniões de *sprint planning*
6.2.2	Utilizar o processo de resolução de problemas de *software*	Resultados do *feedback* devem ser introduzidos no *backlog* do produto por parte do *Product Owner*.

6.2.3	Analisar pedidos de modificação	
6.2.4	Aprovação de pedidos de modificação	Da responsabilidade do *Product Owner*.
6.2.5	Comunicar com utilizadores e reguladores	
6.3	Implementação de modificações	
6.3.1	Usar o processo estabelecido para implementar modificações	Deverão ser construidos novos *epics*(*user stories* para implementação das modificações e introduzidos no *backlog* do produto.
6.3.2	*Re-release* do sistema de *software* modificado	Modificações serão introduzidas na *release* correspondente à *sprint* onde foram implementadas
7	Processo de gestão de risco de *software*	
7.1	Análise de *software* que leva a situações perigosas	
7.1.1	Identificar itens de *software* que podem contribuir para situações perigosas	
7.1.2	Identificar potenciais causas para contribuições para situações perigosas	Cada *user story* é analisada na reunião de *sprint planning*.
7.1.3	Avaliar listas de anomalias publicadas	
7.1.4	Identificar potenciais causas	
7.1.5	Documentar sequências de eventos	Documentação do risco deverá ser introduzida na plataforma de gestão do projeto.
7.2	Medidas de controlo de risco	
7.2.1	Definir medidas de controlo de risco	Cada item de risco identificado deve ser documentado na plataforma de gestão do projeto juntamente com medidas de controlo.
7.2.2	Medidas de controlo de risco implementadas em *software*	Condições de risco devem fazer parte do detalhe das *user stories* e implementadas a cada *sprint*.
7.3	Verificação de medidas de controlo do risco	
7.3.1	Verificar medidas de controlo do risco	Através das *code reviews*.
7.3.2	Documentar qualquer sequência de eventos	Resultado da avaliação de risco feita a cada *sprint* na reunião de *sprint planning*.
7.3.3	Documentar rastreabilidade	Rastreabilidade documentada na plataforma de gestão do projeto e através do sistema de controlo de versões, para código.

7.4	Gestão de risco de mudanças de *software*	
7.4.1	Analisar mudanças ao *software* de dispositivos médicos em relação a segurança	
7.4.2	Analisar o impacto de mudanças de *software* em medidas de controlo de risco existentes	As mudanças são analisadas da mesma forma que os restantes itens do *backlog* do produto. Risco é analisado na reunião de *sprint planning*.
7.4.3	Efetuar atividades de gestão de risco baseadas em análise	
8	Processo de gestão de configurações de *software*	
8.1	Identificação de configurações	
8.1.1	Estabelecer meios para identificar itens de configuração	O sistema de controlo de versões permite a gestão e versionamento de todos os itens de configuração.
8.1.2	Identificar SOUP	Os itens de configuração como código, *scripts* e *tags* podem ser identificados a partir do sistema de controlo de versões. Os restantes (documentação, infraestrutura) deverão ser identificados na CIL (*Configuration Items List*).
8.1.3	Identificar documentação de configuração do sistema	
8.2	Controlo de mudanças	
8.2.1	Aprovar pedidos de modificação	A aprovação de pedidos de mudança é da responsabilidade do *Product Owner*. Itens de configuração serão alterados no decorrer da *sprint* de acordo com as *user stories*.
8.2.2	Implementar mudanças	
8.2.3	Verificar mudanças	Através de *code reviews* ou revisão dos documentos submetidos no sistema de controlo de versões.
8.2.4	Providenciar meios para rastreabilidade de mudanças	Sistema de controlo de versões e ferramenta de gestão do projeto.
8.3	Ter em conta estado de configurações	Sistema de controlo de versões
9	Processo de resolução de problemas de *software*	
9.1	Preparar relatórios de problemas	Ferramenta de gestão do projeto permite catalogar *issues*
9.2	Investigar o problema	Identificação do conteúdo relevante como comentário em cada item na ferramenta de gestão do projeto.

9.3	Aconselhar entidades relevantes	Da responsabilidade do *Product Owner*.
9.4	Usar o processo de controlo de mudanças	Implementações de defeitos deverão ser executadas no decorrer de *sprints* tal como as restantes *stories*.
9.5	Manter registos	Através do sistema de controlo de versões e plataforma de gestão do projeto.
9.6	Analisar problemas para identificar tendências	Da responsabilidade do *Scrum Master*. Tendências deverão ser discutidas na *Sprint Review*.
9.7	Verificar resolução de problemas de *software*	*Code Reviews* e testes automáticos.
9.8	Testar o conteúdo da documentação	Relatórios produzidos pelos testes automáticos ao nível do sistema e regressão.

B

Checklist de auditoria interna para verificação segundo a norma IEC 62304:2006

Para orientação da construção do relatório de não conformidades mencionado na secção 5.1.4, a equipa de QA deve utilizar uma *checklist* de auditoria criada pela organização para o efeito.

Neste anexo encontra-se um exemplo deste documento, orientado para o cumprimento das indicações contidas na norma IEC 62304:2006. Este documento foi construído através de uma análise detalhada à norma em questão e deve ser utilizado da seguinte forma:

1. Cada secção da *checklist* corresponde (não linearmente, mas de uma forma lógica) a um capítulo da norma IEC 62304:2006. O auditor deve, para cada ponto de cada secção, procurar evidências irrefutáveis do cumprimento. Apenas devem ser considerados os pontos correspondentes à classe de segurança do dispositivo. Cada ponto tem a indicação das classes às quais se aplica;

2. Em consequência das evidências encontradas, deve assinalar se o ponto permite cumprimento ou não cumprimento, assim como indicar eventuais sugestões de melhoria e observações;

3. O documento permite fazer uma análise estatística percentual do nível de cumprimento da *sprint* que deve ser analisado de forma a encontrar tendências e melhorar o processo;

4. Com base nesta análise, deve ser produzido um relatório de não-conformidades, que por sua vez deve ser entregue ao *Poduct Owner* para que as mesmas sejam prioritizadas e introduzidas sob a forma de itens de configuração na próxima *sprint*.

OBJECTIVE

AUDIT INFORMATION

PROJECT	<PROJECT CODE>	PROJECT NAME	<PROJECT NAME>
AUDIT	1		
AUDIT TYPE		INTERVIEWEES	
IMS VERSION			
DATE		LEAD AUDITOR	
DURATION (h)		AUDITOR	

PROJECT DETAILS

AGILE PRODUCT OWNER	CUSTOMER
AGILE SCRUM MASTER	LOCATION
START DATE	LIFECYCLE
END DATE	STATE
VCS REPO PATH	

TYPE	TOTAL
COMPLIANT (C)	
NON COMPLIANT (NC)	
NOT APPLICABLE (N/A)	

TYPE	TOTAL
IMPROVEMENT SUGGESTION (IS)	
CORRECTION (CN)	
OBSERVATION (O)	

EXECUTIVE SUMMARY

[Project description]

Non-conformities summary:
-

Corrections:
-

Improvement Suggestions:
-

Observations:
-

ID	AUDIT SCOPE	NC	SCORE
PM	PROJECT MANAGEMENT		
CM	CONFIGURATION MANAGEMENT		
SP	SOFTWARE PLANNING		
RSKM	RISK MANAGEMENT		
VER	VERIFICATION		
DL	DELIVERY		
REQA	REQUIREMENTS ANALYSIS		
SC	SAFETY CLASSIFICATION		
SWD	SOFTWARE DESIGN		
SWT	SOFTWARE TESTING		
SWC	SOFTWARE CONSTRUCTION		

AUDIT SCORE	

ID	SAFETY CLASSIFICATION	C	NC	N/A	IS	CN	O	Standard Items Verified	SC
1	A software safety class is assigned to each system, based on the possible effects of a Hazard							4.3	ABC
2	Safety classification is properly documented							4.3	ABC

COMPLIANCE SCORE:

CHECKLIST SCORE:

ID	PROJECT MANAGEMENT	C	NC	N/A	IS	CN	O	Standard Items Verified	SC
3	**Are actions being managed in na appropriate software management platform?**							n/a	ABC
4	Are the stakeholders identified? Have the relevant roles and responsibilities been defined?							n/a	ABC
5	Are the communication channels and interfaces defined?							n/a	ABC
6	Are the necessary resources identified (e.g. human resources, HW, SW, infrastructure, etc.)?							n/a	ABC
7	Are Lessons Learned analysed during Sprint Retrospectives?							n/a	ABC
8	If developing a product (in Product Development), is there a Product Vision Statement?							n/a	ABC
9	Is the Product Backlog available and updated (described, unique identifiers, prioritized and estimated in story points or hours)?							n/a	ABC
10	Are the top Product Backlog items discussed during Backlog Refinement?							n/a	ABC
11	Are sprints timeboxed?							n/a	ABC
12	Have the Definition of Done and Definition of Ready been defined?							n/a	ABC
13	Are Daily Scrum meetings taking place? Are tasks status and data updates in the Task Board?							n/a	ABC
14	Do Sprint Review meeting occur? Are there records of these meeting?							n/a	ABC
15	Are Sprint Retrospective meetings performed (outputs consist in Sprint Retrospective Book and minutes or issues or tasks in JIRA)?							n/a	ABC
16	Is the Sprint Burndown chart updated for each daily scrum?							n/a	ABC
17	Is Estimation Accuracy monitored? Are actions raised to correct deviations?							n/a	ABC

PROJECT COMPLIANCE SCORE: _____

MANAGEMENT **CHECKLIST SCORE:**

ID	CONFIGURATION MANAGEMENT	C	NC	N/A	IS	CN	O	Standard Items Verified	SC
18	Are the project team members aware of the configuration management strategy?							n/a	ABC
19	Is there a CM plan? Is it updated?							5.1.9	ABC
20	Have the tools that support the software configuration management been described in the Software Plan?							5.1.9	ABC
21	Is there a suitable tagging strategy defined?							8.1.1	ABC
22	Are all software items (e.g. documentation, source code, object or relocatable code, executable code, open source code, files, tools, test software, firmware and data, and all data concerning COTS) subject to configuration management procedures?							8.1.2	ABC
23	Is there a CIL? Is it updated?							8.1.3	ABC
24	Is the title, manufacturer, version and unique identifier clearly documented for all configuration items?							8.1.2	ABC
25	Is there a clear strategy defined for approval change requests?							8.2.1	ABC
26	Is there a verification process in place for every change request?							8.2.3	ABC
27	Are there impact analysis and traceability activities for change requests?							8.2.2; 8.2.4	ABC
28	Is traceability maintained for Epics - Stories - Tasks - Components - Source Code - Tests?							8.2.4; 8.3	ABC

CONFIGURATION MANAGEMENT COMPLIANCE SCORE:

CHECKLIST SCORE:

ID	SOFTWARE PLANNING	C	NC	N/A	IS	CN	O	Standard Items Verified	SC
29	Is a software development plan developed and documented?							5.1.1	ABC
30	Does the software development plan detail the software development lifecycle model to be used?							5.1.1	ABC
31	Does the software development plan datail the development standards, methods and tools available?							5.1.4	ABC
32	Does the software development plan include a testing plan (including integration testing)?							5.1.5	ABC
33	Does the software development plan include a strategy for verification?							5.1.6	ABC
34	Does the software develoment plan include plans for configuration management?							5.1.9	ABC
35	Does the software develoment plan include plans for risk management?							5.1.7	ABC
36	Does the software development plan detail which deliverables will be produced?							5.1.8	ABC
37	Is the software development plan updated when required?							5.1.2	ABC
38	Are Sprint Planning meetings being performed at the beggining of each Sprint?							n/a	ABC

SOFTWARE PLANNING

COMPLIANCE SCORE:

CHECKLIST SCORE:

ID	RISK MANAGEMENT	C	NC	N/A	IS	CN	O	Standard Items Verified	SC
39	Were risks identified and quantified during proposal phase communicated and handed over to the project team?							7.1.1	BC

40	Are project risks identified, monitored and managed?							7.1.2; 7.1.4	BC
41	Are potential causes documented for each risk item?							7.1.2; 7.1.4; 7.1.5	BC
42	Are risk control measures defined and implemented?							7.2.1; 7.2.2	BC
43	Is a verfication strategy documented for every risk item?							7.3.1; 7.3.2	BC
44	Are Hazards traceable to risk items and measures?							7.3.3	BC
45	Are risks ideintified for every change request, as well as impact?							7.4.1; 7.4.2	BC
46	Are risk items being analysed in every Sprint Review meeting?							n/a	BC
47	Are risks being documented and updated at every Sprint Review Meeting?							n/a	BC

RISK MANAGEMENT

COMPLIANCE SCORE:

CHECKLIST SCORE:

ID	VERIFICATION	C	NC	N/A	IS	CN	O	Standard Items Verified	SC
48	Is a verification strategy defined and documented?							5.1.6; 5.5.2	ABC
49	Are code reviews performed for every software unit and tests?							5.5.5	BC
50	Are deliverable documents reviewed?							5.1.8; 5.1.6	ABC
51	Are risk control measures reviewed?							7.3.1	BC

52	Are configuration items reviewed?							8.2.3	ABC

VERIFICATION

COMPLIANCE SCORE:

CHECKLIST SCORE:

ID	DELIVERY	C	NC	N/A	IS	CN	O	NOTES	JIRA
53	Is a Release Strategy defined?							5.8	BC
54	Is the tram ensuring every verification procedure is finished before release?							5.8.1	BC
55	Are all residual anomalies documented and evaluated upon release?							5.8.2; 5.8.3	BC
56	Are the software releases correctly versioned?							5.8.4	ABC
57	Is delivery assured through a continuous delivery strategy?							5.8.5; 5.8.6; 5.8.7; 5.8.8	BC
58	Are deliverables release environments identified in the CIL?							5.8.4	ABC

DELIVERY

COMPLIANCE SCORE:

CHECKLIST SCORE:

ID	REQUIREMENTS ANALYSIS	C	NC	N/A	IS	CN	O	Standard Items Verified	SC
59	Are responsibilities to specify, accept and implement requirements clear amongst team members?							n/a	ABC
60	Are there scope items or high level product roadmap (Epics)?							n/a	ABC
61	Are pending issues tracked and mapped with User Story(s) or Epic(s)? Are issues managed in na appropriate sofrware amanagement tool?							n/a	ABC
62	Is there traceability between User Stories and Epics?							n/a	ABC

63	Do user stories contain: requirement detail; acceptance criteria; context or additional information; assumptions; relation with other User Stories, pending issues, mock-ups; User Interface Design; State Machine entities; Use Case Diagrams?							5.2.2	ABC
64	Have functional and non-functional (Performance, Interface, Operational, Security, Reliability, Maintainability, Usability) requirements been considered?							5.2.2	ABC
65	Are risk control measures included in the software requirements?							5.2.3; 5.2.4	BC
66	Are the requirements updated each Srint, when needed?							5.2.5	ABC
67	Is risk analysis evaluated upon requirements update?							5.2.4; 5.2.6	ABC

REQUIREMENTS ANALYSIS

COMPLIANCE SCORE:

CHECKLIST SCORE:

ID	SOFTWARE TESTING	C	NC	N/A	IS	CN	O	Standard Items Verified	SC
68	Is there a test strategy for the execution of tests describing the tests to be run?							5.1.5; 5.7.1	ABC
69	Are the test goals, testing success criteria (e.g. 100% of tests passed) and requirements coverage defined?							5.7.1	BC
70	Do User Stories contain an acceptance criteria?							5.5.3	BC
71	Are features being retested after change requests?							5.7.3	BC
72	Is testing of non-functional requirements / user stories appropriately captured?							5.7.1	BC
73	If unit testing is applicable, are there unit tests reports?							5.7.5	BC
74	If integration testing is applicable, are there integration tests reports and resulting project incidents recorded?							5.6.7; 5.6.3	BC
75	Is system usability testing performed? Are users involved in usability tests? Are conclusions of the usability tests discussed with the Product Owner?							5.7.5	BC
76	Are test results being recorded? Are there resulting actions?							5.7.5	BC
77	Is there regression testing? Were the tests repeated after correcting bugs in previously tested software?							5.6.6	BC
78	Is there traceability between test cases and User Stories/Requirements?							5.7.5	BC
79	Do test results identify which application version (or configuration item baseline) was tested?							5.7.5	BC

SOFTWARE TESTING

COMPLIANCE SCORE:

CHECKLIST SCORE:

ID	SOFTWARE DESIGN	C	NC	N/A	IS	CN	O	Standard Items Verified	SC
80	Are design standards and conventions applicable to the project scope identified?							5.4.1	BC

81	Is the software architecture refined into smaller software units?							5.3.1	BC
82	Is the detailed design of all software units documented?							5.4.1	BC
83	Are external interfaces, including user interfaces defined and justified?							5.4.3; 5.3.2	C
84	Are non-functional requirements (e.g. performance or security attributes) reviewed associating measurable qualities with use cases/user stories?							5.3.3	C
85	Are Architectural Requirements documented as User Stories?							n/a	ABC
86	Is the system environment including hardware, software, and external systems defined?							5.3.4	BC
87	Is there traceability between User Stories and Architectural Components achievable?							5.3.1	ABC
88	Is High-Level design kept? (Through an appropriate tool like Enterprise Architect)							5.4.1	BC
89	Does the software architecture design contain the software modules and components identified?							5.4.1	BC
90	Is the necessary segregation between components identified for risk control?							5.3.5	BC
91	Is there a strategy described and justified for handling failure of the system (e.g. process termination, system recovery) and for special states (e.g. abnormal termination, error recovery, loosing power)?							n/a	BC
92	Was SAS (Architecture Specification) reviewed?							5.3.6; 5.4.4	BC

SOFTWARE DESIGN

COMPLIANCE SCORE:

CHECKLIST SCORE:

ID	SOFTWARE CONSTRUCTION	C	NC	N/A	IS	CN	O	Standard Items Verified	SC
93	Is there a well defined acceptance criteria for each software unit?							5.5.3	BC

94	Is the project team aware of coding and commenting standards? Is there a report produced by the Continuous Integration tool showing violations of the code standards while making use of static analysis tools?							n/a		ABC
95	Is there a code review strategy? Are there code review reports?							5.5.2		BC
96	Was there a Reuse analysis?							n/a		ABC
97	Is there a unit test strategy? Are there unit test reports?							5.5.5		BC
98	Is there a report produced by the Continuous Integration tool showing the number of passed, failed unit tests and the coverage achieved?							n/a		ABC
99	Is the Continuous Integration tool setup to have automatic deployment to the Quality environment on demand and as a nightly build?							n/a		ABC

SOFTWARE CONSTRUCTION

COMPLIANCE SCORE:

CHECKLIST SCORE:

C

Roteiro das entrevistas realizadas

1. Introdução e contextualização do tema aos entrevistados. Será feita uma introdução à temática, apresentar a motivação da dissertação e detalhados os objetivos da entrevista guiada;
2. Explicação da pertinência da seleção de cada um dos entrevistados e qual é a contribuição que cada um poderá fazer;
3. Explicação da agenda da entrevista e de que forma se processará a mesma.
4. Colocação das questões por categoria. Questões e tópicos apresentados na Tabela C.1.
5. Agradecimento ao(s) participante(s).

Tabela C.1: Categorias e questões efetuadas no decorrer das entrevistas guiadas aos profissionais da área de engenharia.

Tópico	Questões	Categoria
Barreiras para o desenvolvimento ágil em ambiente regulamentado.	Qual são, na experiência profissional do entrevistado, as principais barreiras para o desenvolvimento ágil num ambiente regulamentado?	Genérica
Benefícios das práticas ágeis.	Quais são, na experiência profissional do entrevistado, os principais benefícios das práticas ágeis, em detrimento das práticas mais orientadas à planificação? No contexto apresentado relativamente aos dispositivos médicos, consideram pertinente a implementação de ciclos de vida ágeis para o desenvolvimento de software certificado?	Genérica
Importância do *tailoring* no desenvolvimento de *software* Importância de estrutura e práticas de quality assurance no desenvolvimento de sistemas críticos	O entrevistado considera um tailoring uma prática importante na definição do processo de engenharia? Porquê?	Genérica
Importância de estrutura e práticas de quality assurance no desenvolvimento de sistemas críticos	Na opinião do entrevistado, que benefícios trazem as práticas de quality assurance como reviews e auditorias para o sucesso de um projeto de software?	Genérica
Validação do modelo proposto	Para cada acção de tailoring apresentadas: • Consideram apropriada a acção sugerida? • Na opinião do entrevistado, acham que a acção sugerida colocará overhead adicional para o desenvolvimento da solução?	Genérica

	• Consideram que a acção sugerida é suficiente para garantir evidências de acordo com o contexto de regulamentação?	
Gestão do Risco	Na experiência profissional do entrevistado, qual a importância da gestão do risco para o sucesso de um projeto de software? Que ferramentas são úteis para a construção de evidências a para o processo de gestão do risco?	Gestão de Projeto
Interação com Cliente	Na experiência profissional do entrevistado, qual a importância das características do cliente e dos stakeholders em geral na decisão das metodologias de desenvolvimento.	Gestão de Projeto
Ferramentas	Na experiências do entrevistado, quais são as principais ferramentas que permitem combater as barreiras ao desenvolvimento ágil (e que ajudam com *compliance*)?	Desenvolvimento

Tabela C.2: Categorias e questões efetuadas no decorrer das entrevistas guiadas aos profissionais da dos dispositivos médicos.

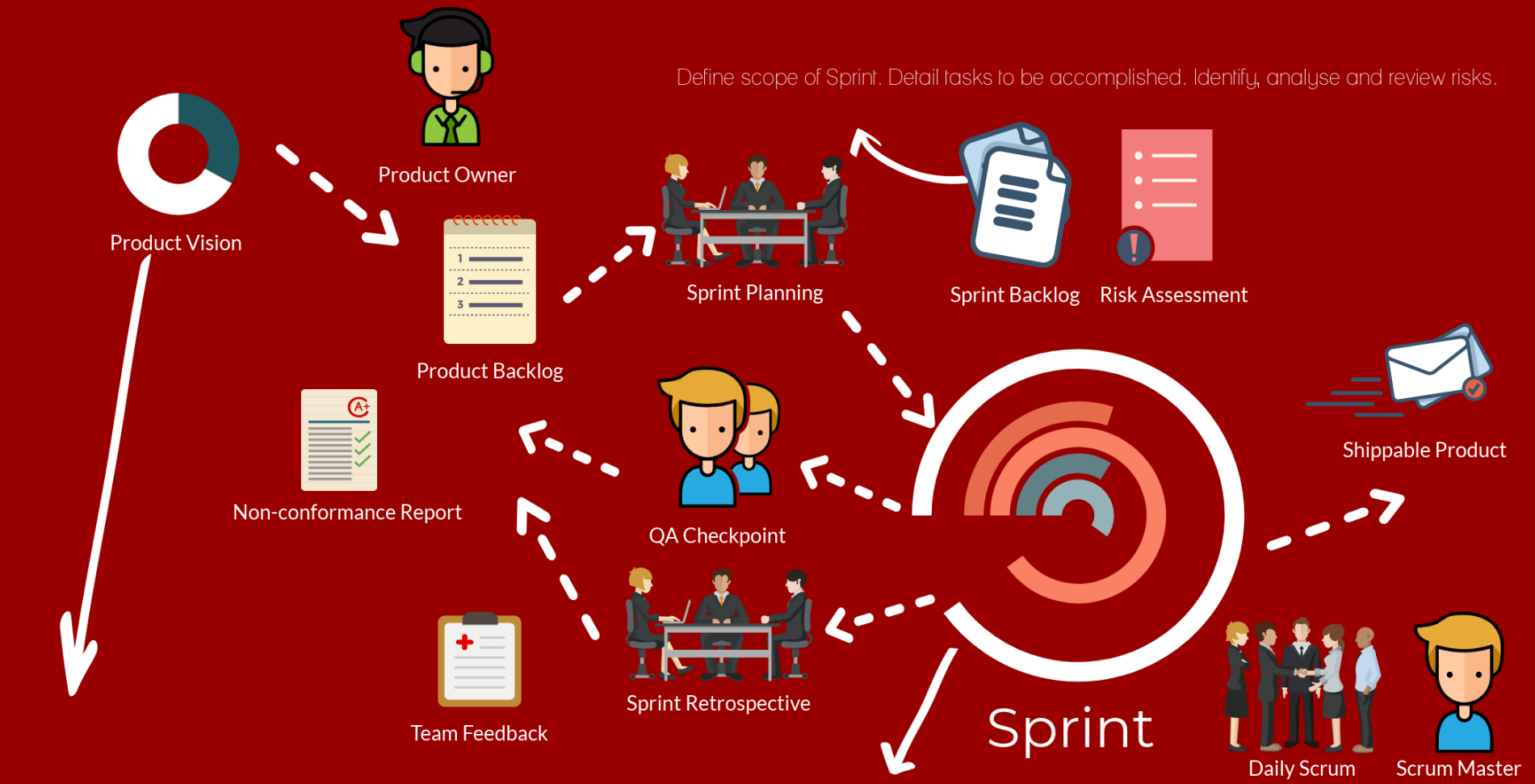
Tópico	Questões
Diretivas da Nova Abordagem.	Esta alteração trouxe melhorias significativas para o panorama dos dispositivos médicos em Portugal?
Aprovação dos Dispositivos Médicos.	Existe, em Portugal, grande interesse em desenvolver dispositivos médicos certificados? Quais são as entidades competentes e de que forma o INFARMED as designa; Pedir ao entrevistado para descrever sucintamente o processo de aprovação e se possível dar algum exemplo;
Alterações ao MDR	Quais são as implicações da alteração ao conceito de finalidade prevista no MDR? A transição dos dispositivos médicos para *software* é uma realidade? Existe necessidade de modificar processos de produção (desenvolvimento de *software*) para adaptação com esta nova realidade? Existem já produtores portugueses a construir software certificado?
Possibilidade de prescrição / comparticipação	É uma possibilidade real? Quais os passos necessários para o conseguir?

D

Agile Fact-sheet

SCRUM for medical devices

Simple tailored implementation of the SCRUM software development life-cycle for compliance according to the IEC 62304:2006 standard



Create an overall vision of the product.
Make sure you create a product backlog, architecture document and planning document with a safety classification for the device

Development iteration. Include all required development, integration and testing activities to have a functional and demonstrable version of the software product at the end of the Sprint.

Roles and Responsabilities

Scrum Master: Ensure the SCRUM life-cycle happens. Teach, train and organize the self-organizing team of developers. Remove impediments and make sure the ceremonies happen without outside interference.

Product Owner: Maximize the value of the developed product. Manage and prioritize the product backlog. Ensure communication and uderstanding with all relevant stakeholders, including approval of change requests. Manage risk and approve risk control measures.

Development Team: Work towards a shippable product at the end of every sprint. The team is self-organizing, meaning that its work should not be disrupted by any outside source.

QA Team: Is not part of the SCRUM Team and is not attached to any specific project. Responsible for auditing the development process at the end of every sprint, as to collect evidences for all items of the IEC 62304:2006 standard.

Ceremonies

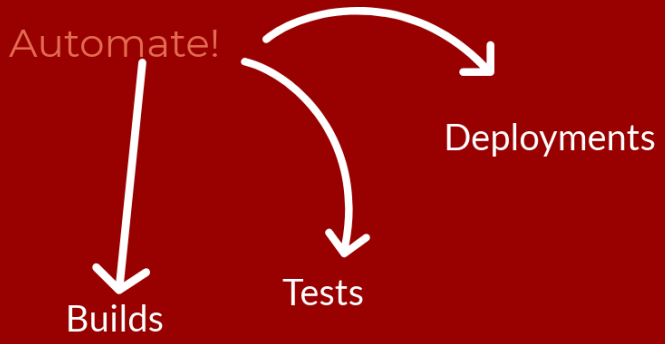
Sprint Planning: Each Sprint starts with a Sprint Planning Meeting. This ceremony has the purpose of defining the scope of the Sprint and detailing the tasks to accomplish that scope. During this meeting, the team must decide on the scope to be done (What to do?), and how they will include that functionality (How to do?). The product owner has the responsibility of presenting an updated and prioritized product backlog, so that the team can accurately estimate each user story to be included in the next sprint (Sprint Backlog). Additionally, during the sprint meeting, the product owner and scrum master should identify the risks concerning the items selected for the sprint. At each planning, these risk should be reevaluated and iterated.

Sprint Review: The Sprint Review Meeting serves the purpose of collaborating on assessing the value of what was done in the Sprint and what are the next things that could be done. This ceremony provides valuable input to the Product Owner to update the Product Backlog and subsequent Sprint Planning meeting.

Sprint Retrospective: The purpose of this meeting is for the Team and the Scrum Master to look back at the Sprint to make the process more effective - delivering the user stories - and more enjoyable - more satisfactory for the Team members. This meeting takes place after the Sprint Review Meeting and before the next Sprint Planning Meeting. The findings of this meeting must be in the form of actionable improvement measures.

Daily Scrum: Daily management of Sprint execution is performed by conducting daily short meetings (15 minutes maximum).

Achieve Compliance



Strategies like Continuous Integration and Deployment are key for building an efficient Configuration Management infrastructure and are able to create tangible evidences of such.
Developers should not waste valuable effort on tasks that can be automated like releases, builds and tests.
Tests should be automated at the unit, integration, system and regression levels and should ideally be able to run at every build cycle. Tests should generate reports of success and failure.

Use Version Control

Version Control software tools are very powerful, not only for making the development team's work easier on a daily basis, but also for providing compliance evidences on several aspects. Firstly, configuration management is assured at a code level, by versioning it and providing the ability to rollback to a certain state of the project. It also provides traceability to tests, if they are commited along with the source code.

Track Issues

Issue tracking software is the team's best friend in a compliant environment. It provides an easy and automatic way for traceability between any kind of configuration item, like risk items, requirements, releases and code. Additionally, if version control is integrated, every commit can originate an issue in the platform.

Verify

Use code reviews for verification whenever possible. Pull requests and branching are excellent strategies for making sure every commit is verified before integration. Documentation can also be commited and follow the same flow. Risk control measures must be verified in the sprint planning meetings.
Sprint 0 is also very important for verification of all outputs of the product vision phase.