Capítulo
1

Introdução

Este capítulo apresenta uma introdução ao assunto de engenharia de software, iniciando pela referência clássica à *crise do software* (Seção 1.1) e aos *mitos do software* (Seção 1.2). Procura-se dar um entendimento às expressões "engenharia de software" (Seção 1.3) e "engenheiro de software" (Seção 1.4), já que há uma grande variedade de definições e entendimentos a respeito destes termos. A *evolução* da área é brevemente apresentada (Seção 1.5), bem como uma classificação dos *tipos de sistemas* referentes à engenharia de software (Seção 1.6), uma vez que este livro se especializa em engenharia de software para sistemas de informação (outros tipos de sistema poderão necessitar de técnicas específicas que não são tratadas aqui).

1.1 A Crise dos Desenvolvedores de Software Menos Preparados

Por algum motivo, os livros de engenharia de software quase sempre iniciam com o tema "crise do software". Essa expressão vem dos anos 1970. Mas o que é isso, afinal? O software está em crise? Parece que não, visto que hoje o software está presente em quase todas as atividades humanas. Mas as *pessoas* que desenvolvem software estão em crise há décadas e, em alguns casos, parecem impotentes para sair dela.

Em grande parte, parece haver desorientação em relação a como planejar e conduzir o processo de desenvolvimento de software. Muitos desenvolvedores concordam que não utilizam um processo adequado e que deveriam investir em algum, mas ao mesmo tempo dizem que não têm tempo ou recursos financeiros para fazê-lo. Essa história se repete há décadas.

A expressão "crise do software" foi usada pela primeira vez com impacto por Dijkstra (1971). Ele avaliava que, considerando o rápido progresso do hardware e das demandas por sistemas cada vez mais complexos, os desenvolvedores simplesmente estavam se perdendo, porque a engenharia de software, na época, era uma disciplina incipiente.

Os problemas relatados por Dijkstra eram os seguintes:

- Projetos que estouram o cronograma.
- Projetos que estouram o orçamento.
- Produto final ineficiente, de baixa qualidade ou que não atende aos requisitos.
- Produtos não gerenciáveis e difíceis de manter e evoluir.
- Produtos que nunca são entregues.

Alguma semelhança com projetos de sistemas do início do Século XXI? Muitas! Sucede que, embora a engenharia de software tenha evoluído como ciência, sua aplicação na prática ainda é limitada.

Mesmo depois de cinquenta anos, ainda são comuns as queixas da alta administração das organizações em relação ao setor de informática quanto a prazos que não são cumpridos, custos ainda muito elevados, sistemas em uso que demandam muita manutenção e também ao fato de que é difícil recrutar profissionais qualificados.

Os usuários também estão infelizes: encontram erros e falhas inadmissíveis em sistemas entregues, sentem-se inseguros em usar tais sistemas e reclamam da constante necessidade de manutenção e do seu alto custo.

Os desenvolvedores, por sua vez, não estão mais satisfeitos: sentem que sua produtividade é baixa em relação ao seu potencial, lamentam a falta de qualidade no produto gerado por seu trabalho, sofrem pressão para cumprir prazos e orçamentos apertados, e ficam inseguros com as mudanças de tecnologia que afetam sua qualificação em relação ao mercado.

Booch (1994) afirma: "*We often call this condition 'the software crisis', but frankly, a malady that has carried on this long must be called 'normal'*"[1]

[1]Frequentemente chamamos essa condição de "crise do software", mas, francamente, uma doença que já dura tanto tempo devia ser chamada de "normalidade".

. Pode-se concluir que a crise do software continuará enquanto os desenvolvedores continuarem a utilizar processos artesanais e a não capitalizar erros e acertos aplicando as modernas técnicas de engenharia de software, muitas das quais descritas neste livro.

Teixeira (2010) compara o desenvolvimento de software ao artesanato da Idade Média, quando, por exemplo, um artesão fazia um par de sapatos como um produto único para cada cliente. Buscava-se a matéria-prima, cortava-se e costurava-se para produzir um sapato que servisse ao cliente. O software, em muitas empresas, ainda é desenvolvido dessa forma artesanal.

Porém, apenas a adoção de processos e padrões efetivamente industriais para a produção de software poderá fazer essa área desenvolver-se mais rapidamente, com mais qualidade e, finalmente, sair dessa dificuldade crônica que já nem pode ser chamada de crise.

1.2 Os Eternos Mitos

São bastante conhecidos também os *mitos* do software, alguns dos quais identificados por Pressman (2005). Esses mitos são crenças tácitas e explícitas que permeiam a cultura de desenvolvimento de software. Os desenvolvedores mais experientes acabam percebendo que estas crenças não têm fundamento, constituindo-se realmente em mitos, mas a cada ano novos desenvolvedores de software entram no mercado e reavivam as velhas crenças, já que seu apelo é grande.

Pressman classifica os mitos em três grupos: *administrativos*, do *cliente* e do *profissional*. Seguem alguns comentários sobre os mitos *administrativos*:

- *A existência de um manual de procedimentos e padrões é suficiente para a equipe produzir com qualidade.* Na verdade, deve-se questionar se o manual é realmente usado, se ele é completo e atualizado. Muitas vezes, manuais são ignorados por serem incompletos, irrelevantes, imprecisos ou incompreensíveis; outras vezes eles são ignorados simplesmente porque não foram efetivamente adotados pela equipe. Deve-se trabalhar com processos e padrões que possam ser gerenciáveis e otimizados, ou seja, sempre que a equipe identificar falhas no processo, deve ser possível modificá-lo.
- *A empresa deve produzir com qualidade, pois tem ferramentas e computadores de última geração.* Na verdade, ferramentas e computadores de boa qualidade são condições necessárias, mas não suficientes. Parafraseando Larman (2001), comprar uma ferramenta não transforma você instantaneamente em arquiteto. Além de possuir boas ferramentas, a equipe deve possuir bons conhecimentos sobre o uso destas.
- *Se o projeto estiver atrasado, sempre é possível adicionar mais programadores para cumprir o cronograma*. Embora este mito até pareça intuitivo, o problema é que pessoas que desenvolvem software juntas precisam se comunicar, e cada vez que novos membros são adicionados à equipe, novos canais de comunicação surgem, sobrecarregando a todos. O desenvolvimento de software é uma tarefa altamente complexa. Adicionar mais pessoas sem que haja um planejamento prévio pode causar mais atrasos. Se não fosse assim, um programa de 20 mil linhas poderia ser escrito rapidamente por 20 mil programadores.
- *Um bom gerente pode gerenciar qualquer projeto*. Gerenciar não é fazer, e o desenvolvimento de software é um processo complexo por vários motivos. Assim, mesmo que o gerente seja competente, se não houver na equipe competência técnica, boa comunicação e um processo de trabalho previsível, ele pouco poderá fazer para obter um produto com a qualidade desejada.

Entre os mitos relacionados ao *cliente*, pode-se citar:

- *Uma declaração geral de objetivos é suficiente para iniciar a fase de programação. Os detalhes podem ser adicionados depois.* É verdade que não se pode esperar que a especificação inicial do sistema esteja correta e completa antes de se iniciar a programação, mas ter isso como meta é péssimo. Deve-se procurar obter o máximo de detalhes possível antes de iniciar a construção

do sistema. Técnicas mais sofisticadas de análise de requisitos e uma equipe bem treinada poderão ajudar a construir as melhores especificações possíveis sem perda de tempo.

- *Os requisitos mudam com frequência, mas sempre é possível acomodá-los, pois o software é flexível*. Na verdade, o *código* é fácil de mudar – basta usar um editor. Mas mudar o código sem introduzir erros é uma tarefa bastante improvável, especialmente em organizações com baixa maturidade de processo. O software só será efetivamente flexível se for construído com esse fim. É necessário, entre outras coisas, identificar os requisitos permanentes e os transitórios, e, no caso dos transitórios, preparar o sistema para sua mudança utilizando padrões de *design* adequados. Mesmo que o software não seja um elemento físico, como um edifício ou uma ponte (mais difíceis de serem modificados), a mudança do software também implica esforço e custo (em tempo e dinheiro), e muitas vezes esse esforço e esse custo não são triviais.
- *Eu sei do que preciso.* Os desenvolvedores costumam dizer o inverso: o cliente *não sabe* do que precisa. É necessário que os analistas entendam que os clientes (a não ser que sejam técnicos especializados) raramente sabem do que realmente precisam e têm grande dificuldade para descrever e até mesmo para lembrar-se de suas necessidades. De outro lado, analistas muitas vezes confundem as *necessidades do cliente* (alvo da *análise*) com as *soluções possíveis* (alvo do *design*). Por exemplo, um analista pode achar que o cliente precisa de um sistema *web* acessando um banco de dados relacional, mas essa não é uma boa descrição de uma necessidade. É a descrição de uma solução para uma necessidade que possivelmente não foi estabelecida de maneira clara. Outras soluções diferentes de *web* e banco de dados poderiam ser aplicadas.

Outros mitos de Pressman dizem respeito ao *profissional*, e são comentados a seguir:

- *Assim que o programa for colocado em operação, nosso trabalho terminou*. Na verdade, ainda haverá *muito* esforço a ser despendido depois da instalação do sistema, por causa de erros dos mais diversos tipos. Estudos indicam que mais da metade do esforço despendido com um sistema ocorre depois de sua implantação (Von Mayrhauser & Vans, 1995).
- *Enquanto o programa não estiver funcionando, não será possível avaliar sua qualidade*. Na verdade, o programa é apenas um dos artefatos produzidos no processo de construção do software (possivelmente o mais importante, mas não o único). Existem formas de avaliar a qualidade de artefatos intermediários como casos de uso e modelos conceituais para verificar se estão adequados mesmo antes da implementação do sistema. Além disso, código de programa pode ter suas qualidades internas avaliadas independentemente de estar funcionando ou não. Existem inclusive sistemas hoje que fazem essa verificação automaticamente, como por exemplo o Sonar (confira no QR Code ao lado). [QRC 1.1]

- *Se eu esquecer alguma coisa, posso arrumar depois*. Quanto mais o processo de desenvolvimento avança, mais caras ficam as modificações em termos de tempo e dinheiro.
- *A única entrega importante em um projeto de software é o software funcionando*. Talvez essa seja a entrega mais importante, porém, se os usuários não conseguirem utilizar o sistema pouco valor ele terá. O final de um projeto de informática costuma envolver mais do que simplesmente entregar o software na portaria da organização do cliente. É necessário realizar testes de operação, importar dados, treinar usuários, definir procedimentos operacionais. Assim, outros artefatos, além do software funcionando, poderão ser necessários.

Leveson (1995) também apresenta um conjunto de mitos correntes, a maioria dos quais está relacionada à confiabilidade do software:

- *O teste do software ou sua verificação formal pode remover todos os erros*. Na verdade, o software é construído com base em uma especificação de requisitos que usualmente é feita em linguagem natural e, portanto, aberta a interpretações. Nessas interpretações pode haver erros ocultos. Além disso, a complexidade do software contemporâneo é tão grande que se torna in-

viável especificar formalmente ou mesmo testar todas as possibilidades de uso. Assim, a melhor opção consiste em testar os caminhos mais representativos ou capazes de provocar erros. As técnicas de teste é que vão indicar quais caminhos é interessante verificar.

- *Aumentar a confiabilidade do software aumenta a segurança*. O problema é que o software pode ser confiável apenas em relação à sua especificação, ou seja, ele pode estar *fazendo certo a coisa errada*. Isso normalmente se deve a requisitos mal compreendidos.
- *O reuso de software aumenta a segurança*. Possivelmente os componentes reusados já foram testados, mas o problema é saber se os requisitos foram corretamente estabelecidos, o que leva de volta ao mito anterior. Além disso, se os componentes reusados forem sensíveis ao contexto (o que acontece muitas vezes), coisas inesperadas poderão acontecer.

Agora, de nada adianta apenas estar consciente dos mitos. Para produzir software com mais qualidade e confiabilidade é necessário utilizar uma série de conceitos e práticas. Ao longo deste livro, vários conceitos e práticas úteis em Engenharia de Software serão apresentados ao leitor de forma que ele possa compreender o alcance dessa área e seu potencial para a melhoria dos processos de produção de sistemas.

1.3 (In)Definição de Engenharia de Software

Segundo a Desciclopédia, a Engenharia de Software pode ser definida assim: "*A Engenharia de Software forma um aglomerado de conceitos que dizem absolutamente nada e que geram no estudante dessa área um sentimento de 'Nossa, li 15 kg de livros desta matéria e não aprendi nada'. É tudo bom senso.*"

Apesar de considerar que o material publicado na Desciclopédia seja de caráter humorístico, a sensação que muitas vezes se tem da Engenharia de Software é mais ou menos essa. Pelo menos foi essa a sensação do autor deste livro e de muitos de seus colegas no passado.

Efetivamente, não é algo simples conceituar e praticar a Engenharia de Software. Mas é *necessário*. Primeiramente, deve-se ter em mente que os processos de Engenharia de Software são diferentes, dependendo do tipo de software que se vai desenvolver. O Capítulo 3, por exemplo, vai mostrar que, dependendo do nível de conhecimento ou estabilidade dos requisitos, deve-se optar por diferentes características para o modelo de desenvolvimento. O Capítulo 8, por outro lado, vai mostrar que uma área aparentemente tão subjetiva como "riscos" pode ser sistematizada e tratada efetivamente como um processo de engenharia, e não como adivinhação. O Capítulo 7 apresentará formas objetivas e padronizadas para mensurar o esforço do desenvolvimento de software, de forma a gerar números que sejam efetivamente realistas, o que já vem sendo comprovado em diversas empresas.

Assim, espera-se que este livro consiga deixar no leitor a sensação de que a Engenharia de Software é possível e viável, desde que se compreenda em que ela realmente consiste e como pode ser utilizada na prática.

Várias definições de Engenharia de Software podem ser encontradas na literatura, mas neste livro será considerada com maior ênfase a definição da Engenharia de Software como *o processo de estudar, criar e otimizar os processos de trabalho para os desenvolvedores de software*. Assim, embora isso não seja consenso geral, considera-se, que as atividades de levantamento de requisitos, modelagem, *design*[2]

[2]As palavras inglesas *design* e *project* costumam ser traduzidas para o português como "projeto". Para evitar confusão entre os dois significados, neste livro o termo *design* não será traduzido.

e codificação, por exemplo, não são típicas de um engenheiro de software, embora, muitas vezes, ele seja habilitado a realizá-las. Sua tarefa consiste mais em observar, avaliar, orientar e alterar os processos produtivos quando necessário. Assim, a rigor, as atividades de análise, *design*, programação e teste de software, e mesmo as atividades de gerenciamento do processo de desenvolvimento, seriam realizadas por profissionais identificados como analistas, *designers*, programadores, testadores e gerentes, respectivamente, cabendo ao engenheiro de software a definição, avaliação e acompanhamento de seus processos produtivos. A seção seguinte procura deixar mais clara essa distinção

entre as atividades dos desenvolvedores e do engenheiro de software.

1.4 O Engenheiro de Software

Uma das primeiras confusões que se faz nesta área é entre o *desenvolvedor* e o *engenheiro* de software. Isso equivale a confundir o engenheiro civil com o pedreiro ou com o mestre de obras.

O desenvolvedor, seja ele analista, *designer*, programador ou gerente de projeto, é um *executor* do processo de construção de software. Os desenvolvedores, de acordo com seus papéis, têm a responsabilidade de descobrir os requisitos e transformá-los em um produto executável. Mas o engenheiro de software tem um meta papel em relação a isso. Pode-se dizer que o engenheiro de software não coloca a mão na massa, assim como o engenheiro civil não vai à obra assentar tijolos ou concretar uma laje. Então, o engenheiro de software não é um desenvolvedor que trabalha nas atividades de análise e produção de código.

Porém, a comparação com a engenharia civil termina por aqui, já que o engenheiro civil será o responsável pelo projeto físico da obra. Na área de Computação, a especificação do projeto lógico e físico fica a cargo do analista e do *designer*, o primeiro com a responsabilidade de identificar e modelar os requisitos e o segundo de desenhar uma solução que utilize a tecnologia para transformar esses requisitos em um sistema executável. No âmbito do software, há muito tempo esses papéis também têm sido confundidos com os do engenheiro de software.

Pode-se afirmar que o engenheiro de software assemelha-se mais ao engenheiro de produção. Ele deve fornecer aos desenvolvedores (inclusive gerentes, analistas e *designers*) as ferramentas e processos que deverão ser usados e será o responsável por verificar se esse uso está sendo feito efetivamente e de forma otimizada. Além disso, caso tais ferramentas e processos apresentem qualquer problema, ele será o responsável por realizar as modificações necessárias, garantindo assim sua contínua melhoria.

O engenheiro de software, portanto, não desenvolve nem especifica software. Ele viabiliza e acompanha o processo de produção, fornecendo e avaliando as ferramentas e técnicas que julgar mais adequadas a cada projeto ou empresa.

É necessário, ainda, distinguir o engenheiro de software do gerente de projeto. O gerente de projeto deve planejar e garantir que este seja executado de forma adequada dentro dos prazos e orçamento especificados. Mas o gerente de projeto tem uma responsabilidade mais restrita ao projeto em si, e não ao processo de produção. Nesse sentido, ele também é um executor. Ele utiliza as disciplinas definidas no processo de engenharia de software para gerenciar seu projeto específico, mas não é necessariamente o responsável pela evolução desses processos, nem necessariamente o responsável por sua escolha. Esse papel cabe ao engenheiro de software.

Em equipes pequenas, uma mesma pessoa poderá atuar no papel de engenheiro de software e simultaneamente em outros papeis, como analista ou gerente. Aqui não se está falando necessariamente de pessoas diferentes para cada papel, mas de atribuições distintas que poderão estar alocadas a uma mesma pessoa ou a pessoas diferentes. Resumindo, os diferentes papéis poderiam ser caracterizados assim:

- O *engenheiro de software* escolhe e, muitas vezes, especifica os processos de planejamento, gerência e produção a serem implementados. Ele acompanha e avalia o desenvolvimento de todos os projetos da empresa para verificar se o processo estabelecido é executado de forma eficiente e efetiva. Caso sejam necessárias mudanças no processo estabelecido, ele as identifica e realiza, garantindo que a equipe adote tais mudanças. Ele reavalia o processo continuamente.
- O *gerente de projeto* cuida de um projeto específico, garantindo que os prazos, orçamento, escopo e objetivos de qualidade sejam cumpridos em relação ao produto a ser desenvolvido. Ele segue as práticas definidas no processo de engenharia. Ele também é responsável por verificar a aplicação do processo pelos desenvolvedores e, se necessário, reporta-se ao engenheiro de software para sugerir melhorias.
- O *analista*, em sua definição mais geral, é um desenvolvedor responsável pela compreensão do

problema relacionado ao sistema que se deve desenvolver, ou seja, pelo levantamento dos requisitos e sua efetiva modelagem. O analista deve, portanto, descobrir o que o cliente precisa (por exemplo, controlar suas vendas, comissões, produtos etc.).

- O *designer* deve levar em conta as especificações do analista e propor a melhor tecnologia para produzir um sistema executável para elas. Deve, então, apresentar uma solução para as necessidades do cliente (por exemplo, propor uma solução baseada em *web*, com um banco de dados centralizado acessível por dispositivos móveis etc.). Algumas vezes o termo "*designer*" é associado apenas aos aspectos de interface gráfica do sistema, mas pode-se ter designers também atuando na arquitetura interna do software.
- O *programador* vai construir a solução física a partir das especificações do *designer*. É ele quem gera o produto final, e deve conhecer profundamente a linguagem e o ambiente de programação, bem como as bibliotecas que for usar, além de ter algum conhecimento sobre teste e depuração de software.

Como já comentado, em algumas organizações essa divisão de papéis pode não ser observada estritamente. Usualmente, apenas organizações de médio e grande porte podem se dar ao luxo de ter um ou mais engenheiros de software dedicados. Mas é importante que se tenha em mente que, ainda que uma pessoa execute mais de um papel nesse processo, essas atribuições são especialidades distintas na qual um profissional pode se aprofundar cada vez mais.

Há outros papeis também não menos importantes como o de testador, arquiteto de software, gerente de qualidade, analista de negócio etc. Vários destes papeis são apresentados e comentados ao longo do livro.

1.5 Evolução da Engenharia de Software

A maioria dos primeiros computadores, construídos entre as décadas de 1930 e 1940, não possuíam software: os comandos eram implantados na máquina a partir de conexões físicas entre os componentes. À medida que se percebeu a necessidade de computadores mais flexíveis, surgiu o software, que consiste em um conjunto de instruções que fazem a máquina produzir algum tipo de processamento. Como o software é um construto abstrato, sua produção não se encaixava perfeitamente em nenhuma das engenharias, nem mesmo na mecânica e na elétrica, que são as mais próximas, por terem relação com as máquinas que efetuam as computações. Surgiu, então, o conceito de *engenharia de software*, inicialmente referindo-se aos processos para a produção desse tipo de construto abstrato.

Aceita-se que a primeira conferência sobre Engenharia de Software tenha sido a Conferência de Engenharia de Software da OTAN, organizada em Garmish, Alemanha, em 1968 (Bauer, 1968). Apesar disso, o termo já era usado desde os anos 1950.

O período da década de 1960 até meados da década de 1980 foi marcado pela chamada “crise do software”, durante a qual foram identificados os maiores problemas relacionados à produção de software, especialmente em larga escala. Inicialmente, a crise referenciava especialmente questões relacionadas com orçamento e cronograma de desenvolvimento, mas posteriormente passou também a abranger aspectos de qualidade de software, uma vez que os sistemas, depois de prontos, apresentavam muitos problemas, causando prejuízos.

Segundo Edsger Dijkstra, um dos responsáveis pelo uso do termo "crise do software", a coisa ia bem enquanto as máquinas de computação não possuíam mecanismos de interrupção por hardware. Antes disso, os programas eram bastante previsíveis e usualmente o que se planejava era executado *ipsis litteris*. Porém, com a introdução dos mecanismos de interrupção, especialmente para permitir o uso do computador por várias aplicações ao mesmo tempo, o comportamento de um programa passou a ser bem mais imprevisível.

Um exemplo clássico da crise de software dos anos 1960 foi o projeto do sistema operacional OS/360, que utilizou mais de mil programadores. Brooks (1975) afirmou ter cometido um erro que custou milhões à IBM nesse projeto, por não ter definido uma arquitetura estável antes de iniciar o

desenvolvimento propriamente dito. Atualmente, a *Lei de Brooks* afirma que adicionar programadores a um projeto atrasado faz com que ele fique ainda mais atrasado.

Por décadas, a atividade de pesquisa tentou resolver a crise do software. Cada nova abordagem era apontada como uma "bala de prata" para solucionar a crise. Porém, pouco a pouco, chegou-se ao consenso de que tal solução mágica não existia. Ferramentas *CASE* (*Computer Aided Software Engineering*), especificação formal, prototipação, processos, componentes, teste automatizado etc. foram boas técnicas que ajudaram a engenharia de software a evoluir, mas hoje não se acredita mais em uma solução única e salvadora para os complexos problemas envolvidos com a produção de software.

Os anos 1990 presenciaram o surgimento da Internet e a consolidação da orientação a objetos como o paradigma predominante na produção de software. A mudança de paradigma e o *boom* da Internet mudaram de forma determinante a maneira como o software era produzido. Novas necessidades surgiram e sistemas cada vez mais complexos, acessíveis de qualquer lugar do mundo, substituíram os antigos sistemas *stand-alone*. Com isso, novas preocupações relacionadas à segurança da informação e à proliferação de vírus e *spam* surgiram e passaram a fazer parte da agenda dos desenvolvedores de software.

Nos anos 2000, o crescimento da demanda por software em organizações de pequeno e médio porte levou ao surgimento de soluções mais simples e efetivas para o desenvolvimento de software para essas organizações. Assim surgiram os *métodos ágeis*, que procuram desburocratizar o processo de desenvolvimento e deixá-lo mais adequado a equipes pequenas mas competentes, capazes de desenvolver sistemas sem a necessidade de extensas listas de procedimentos ou de "receitas de bolo".

Atualmente, a área vem se estabelecendo como um corpo de conhecimentos coeso. O surgimento do *SWEBOK* e sua adoção como padrão internacional em 2006 (ISO/IEC TR 19759) foi um avanço para a sistematização do corpo de conhecimentos da área. Uma versão atualizada do SWEBOK, conhecida com SWBOK 3 foi lançada em 2013 e a partir de 2016 a IEEE Computer Society iniciou um processo de evolução para o desenvolvimento de futuras versões do documento.

1.6 Tipos de Software do Ponto de Vista da Engenharia

Não existe um processo único e ideal para desenvolvimento de software, porque cada sistema tem suas particularidades. Porém, usualmente, podem-se agrupar os sistemas de acordo com certas características e então definir modelos de processo mais adequados a elas. Do ponto de vista da engenharia de software, os sistemas podem ser classificados da seguinte forma:

- *Software básico*: são os compiladores, *drivers* e componentes do sistema operacional. Normalmente, usuários que não sejam profissionais de computação nem tomam conhecimento da existência deste tipo de software.
- *Software de tempo real*: são os sistemas que monitoram, analisam e controlam eventos do mundo real. Este tipo de sistema é bastante comum na indústria e em dispositivos eletromecânicos existentes em veículos e máquinas computadorizadas em geral.
- *Software comercial*: são os sistemas aplicados nas empresas, como controle de estoque, vendas etc. Tais sistemas usualmente acessam bancos de dados. São também referenciados usualmente como *sistemas de informação* já que o interesse destas empresas e organizações normalmente está em manter controle sobre a informação que elas produzem e consomem.
- *Software científico e de engenharia*: são os sistemas que utilizam intenso processamento de números. Trata-se, por exemplo, de ferramentas de desenho técnico ou de cálculo científico para as mais variadas áreas do conhecimento.
- *Software embutido ou embarcado*: são os sistemas de software presentes em *smartphones*, eletrodomésticos, veículos, *drones* etc. Normalmente, tais sistemas precisam trabalhar sob severas restrições de espaço, tempo de processamento e gasto de energia. Muitas vezes, embora nem

sempre, eles podem ter também restrições para trabalhar em tempo real.

- *Software pessoal*: são os sistemas usados por pessoas no dia a dia, como processadores de texto, planilhas etc. A partir do surgimento da cultura de *smartphones*, esse tipo de software também vem sendo conhecido genericamente como "*aplicativos*".
- *Jogos*: embora existam alguns jogos cujo processamento não é muito complexo, existem também aqueles que exigem o máximo dos computadores em função da qualidade de gráficos e da necessidade de reação em tempo real. Apesar disso, todas as categorias de jogos têm características intrínsecas que extrapolam o domínio da engenharia de software. Jogos usualmente são produzidos por equipes multidisciplinares envolvendo profissionais de computação, designers, artistas, roteiristas. Além disso, a indústria de jogos tem desenvolvido suas próprias ferramentas de produção, as assim chamadas "*engines*", que permitem simplificar significativamente o esforço de programação.
- *Inteligência artificial* (*IA*): são os sistemas especialistas, redes neurais e sistemas capazes de alguma forma de aprendizado. Além de serem sistemas independentes, com um tipo de processo de construção próprio, podem também ser embutidos em outros sistemas. Embora a IA tenha passado por altos e baixos ao longo da história da computação, atualmente o investimento mundial nesta forma de software é bastante significativa já que cada vez mais seus resultados têm sido transformados em produtos úteis. Em grande parte, isso foi possível não só pela evolução das técnicas de IA em si, mas também pela gigantesca disponibilidade atual de dados de todos os tipos colhidos por bilhões de dispositivos computacionais espalhados pelo mundo.

Essa classificação, porém, não é completa, detalhada nem exaustiva. Ela apenas ilustra os diferentes tipos de sistemas que são desenvolvidos com o uso de software e, eventualmente, de hardware.

Neste livro, a ênfase está nos *sistemas de informação*, ou seja, será mostrado basicamente como desenvolver um processo de engenharia de software para sistemas do tipo “comercial” para uso em empresas ou organizações. Apesar disso, algumas dessas técnicas poderão ser aplicadas também ao desenvolvimento de outros tipos de software, eventualmente.

FIZ UM ÍNDICE REMISSIVO EM CADA CAPITULO PARA FACILITAR A LOCALIZAÇÃO DAS PALAVRAS. A IDEIA É JUNTAR TODOS EM UM REMISSIVO GERAL DO LIVRO NO FINAL

REMISSIVO DO CAPÍTULO