Renato Barbieri

Uma Breve História da Agilidade

Uma Breve História da Agilidade

Renato Barbieri

Uma Breve História da Agilidade

Projeto gráfico e diagramação: Renato Barbieri

Edição de texto: Pilar Sanchez Albaladejo

Capa criada pelo autor usando a plataforma Shutterstock (https://www.shutterstock.com/).

Foto da capa: Mountains in Little Cottonwood Canyon, Utah — © jfunk / Adobe Stock.

Foto do tecladista Rick Wakeman gentilmente cedida por seu autor Lee Wilkinson.

Imagem do contrato assinado por Alistair Cockburn com o hotel em Snowbird (figura 4.1) e o rascunho dos princípios da Agilidade (figura 4.4) foram gentilmente cedidos por Alistair Cockburn.

Cartuns do Comic Agilé, incluídos nos capítulos 1 (figura 1.2), 5 (figura 5.9) e 6 (figura 6.5), foram gentilmente cedidos por Luxshan Ratnaravi e Mikkel Noe-Nygaard.

Barbieri, Renato
Uma breve história da agilidade [livro eletrônico] / Renato Barbieri. -- São Paulo: Ed. do Autor, 2024.
PDF

Bibliografia.
ISBN 978-65-01-01712-9

1. Administração - Metodologia 2. Engenharia de software 3. Software - Desenvolvimento 4. Tecnologia da informação - Administração I. Título.

CDD-004

Índices para catálogo sistemático:

1. Tecnologia da informação : Ciências da computação 004

A todos os autores que nunca se conformaram em aceitar o status quo.

A todos os leitores sempre curiosos e abertos a novos aprendizados.

Sumário

1. Introdução

“Aqueles que não podem lembrar o passado
estão condenados a repeti-lo.”

George Santayana, filósofo espanhol*

* George Santayana é um pseudônimo de Jorge Augustín Nicolás Ruiz de Santayana y Borrás, um filósofo, poeta e ensaísta espanhol. O aforismo acima aparece em “A Vida da Razão”, livro publicado em 1905.

É Novidade para Mim!

A música sempre foi minha paixão. Sempre considerei que descobri o piano um pouco tarde: eu tinha 12 anos, em 1973, quando comecei a dedilhar as primeiras melodias no piano de casa. Dizem que não existe isso de começar tarde, e é bem verdade que encontrei muita gente que me contava ter tido aulas de piano quando eram bem pequenos, mas logo desistiram porque não tinham paciência para praticar, e queriam mesmo brincar com seus carrinhos ou suas bonecas. Olhando para trás, acho que tive muita sorte, pois a década de 1970 foi muito rica para os amantes do rock, e hoje acredito que me interessei por música na hora certa. Meu primeiro ídolo foi o grande tecladista britânico Rick Wakeman. De formação clássica, ele foi um dos pioneiros no uso de sintetizadores no rock, mas para mim, ele brilhava mesmo ao piano. Além de ser um virtuoso, é um grande contador de histórias, e muito engraçado também.

Recentemente, li uma entrevista[1] que ele deu para a revista Louder, comentando sobre sua turnê de 2023, onde resgata suas primeiras obras. Ao ser indagado pelo entrevistador de onde vinha essa onda nostálgica, ele contou a seguinte história:

> *"Um tempo atrás aprendi uma lição importante com um garoto brasileiro. Ele estava no saguão do hotel onde eu estava hospedado, com uma cópia do meu álbum Six Wives (meu primeiro sucesso solo, de 1973) e queria que eu o autografasse para ele. Disse ter 16 anos. Foi quando perguntei: 'Por que você gosta dessas músicas antigas?', ele me pareceu contrariado e disse: 'Sr. Wakeman, pode ser música antiga para você, mas é tudo novo para mim. Só fiquei conhecendo este seu trabalho há duas semanas. Você precisa se lembrar de que, quando está no palco tocando essas músicas, algumas pessoas estão ouvindo aquilo pela primeira vez.' Confesso que isso me deu o que pensar."*

1. "That was real food for thought": Rick Wakeman on the surprise lesson he learned from a 16-year-old Brazilian fan, Louder, 5 de fevereiro de 2024, https://www.loudersound.com/features/rick-wakeman-tour-interview

Figura 1.1 Rick Wakeman cercado por teclados em 2017.

Fonte: Foto por © Lee Wilkinson 2017 - www.lwmultimedia.co.uk

Como o meu ídolo da adolescência, isso também me deu o que pensar. A palavra “ágil” está associada ao desenvolvimento de software apenas desde 2001, e muitos dos meus alunos nasceram depois disso. Para muitos deles, essa será a primeira vez que terão a oportunidade de ler sobre a origem e a história da Agilidade em português. Encontrei assim motivação suficiente para assumir esse trabalho.

Iniciei minha carreira em software no final da década de 1980, e vivi as várias evoluções e revoluções no mundo da tecnologia da informação, desde então. Imagino que tenha condições de contar um pouco da minha visão dos fatos antes e depois do advento da Agilidade, e compartilhar aquilo que aprendi sobre o movimento, com o qual estou envolvido desde 2002.

Minha intenção é resgatar pontos importantes dessa história, pois muitos livros sobre Agilidade nunca foram traduzidos para

português, e acabamos contando apenas com uma pequena e restrita amostra do universo da Agilidade. Sempre senti essa carência ao indicar livros para os meus alunos e colegas que demonstravam interesse em aprofundar seus conhecimentos de Agilidade, inclusive sobre as origens do movimento, mas sem o domínio do idioma inglês.

Acredito que essa minha iniciativa seja particularmente importante atualmente, quando a Agilidade é questionada por tantos, e quando existe tanta confusão sobre os conceitos básicos dessa filosofia. Soma-se a isso a industrialização, a "comoditização" da Agilidade e temos uma banalização generalizada dos conceitos originais.

É preciso resgatar a essência do movimento e torná-la acessível aos lusófonos como eu e você. Para isso, vamos primeiro precisar viajar um pouco no tempo e viver o contexto que nos trouxe até aqui. No meio dessa viagem, vamos encontrar alguns bons samaritanos que dedicaram seus esforços na melhoria da atividade de desenvolvimento de software, e descobrir como um punhado deles inadvertidamente mudou o mundo.

Guerra de Metodologias 2.0?

Hoje, vivemos uma nova "guerra de metodologias", a exemplo daquela que vivi nos anos 1990, sendo que, paradoxalmente, as abordagens Ágeis deveriam ser vistas como alternativas complementares, abertas, adaptáveis. Será que não aprendemos nada nos últimos 50 anos? Sempre provoquei meus alunos com a seguinte frase: "Se estão aqui para aprender a 'metodologia Ágil', então estão na sala errada — porque não existe **A** metodologia Ágil. Existem abordagens Ágeis que, através da experimentação e adaptação, podem nos auxiliar a encontrarmos a **nossa** maneira de desenvolver software."

Henrik Kniberg, em publicação no seu blog sob o título: "Scrum and XP fit together"[2], exemplifica muito bem essa ideia ao discutir a união de abordagens complementares como o Scrum e o Extreme Programming:

> *"Jeff Sutherland e Ron Jeffries concordam que é uma boa ideia começar com o Scrum e permitir que o XP siga naturalmente (pois, equipes Scrum são autogerenciadas e podem escolher quaisquer métodos de engenharia que desejarem.)*
>
> *"Mike Cohn e Robert C Martin acreditam (ou esperam) que estas duas abordagens irão se fundir em algum tipo de 'Ágil genérico'. Mike é da opinião de que será simplesmente conhecida como 'a maneira como desenvolvemos software'. (Baseado em discussões pessoais com os dois)."*

Em seu curso Deep Agile (2007), Jeff e Ron comentavam que não se vê equipes Scrum de alto desempenho sem as práticas de engenharia do XP. E que é difícil escalar equipes XP sem o Scrum, dado que este resolve os problemas de interface do XP com a liderança.

2. "Scrum and XP fit together", Henrik Kniberg, 13 de outubro de 2007, https://blog.crisp.se/2007/10/13/henrikkniberg/1192249140000

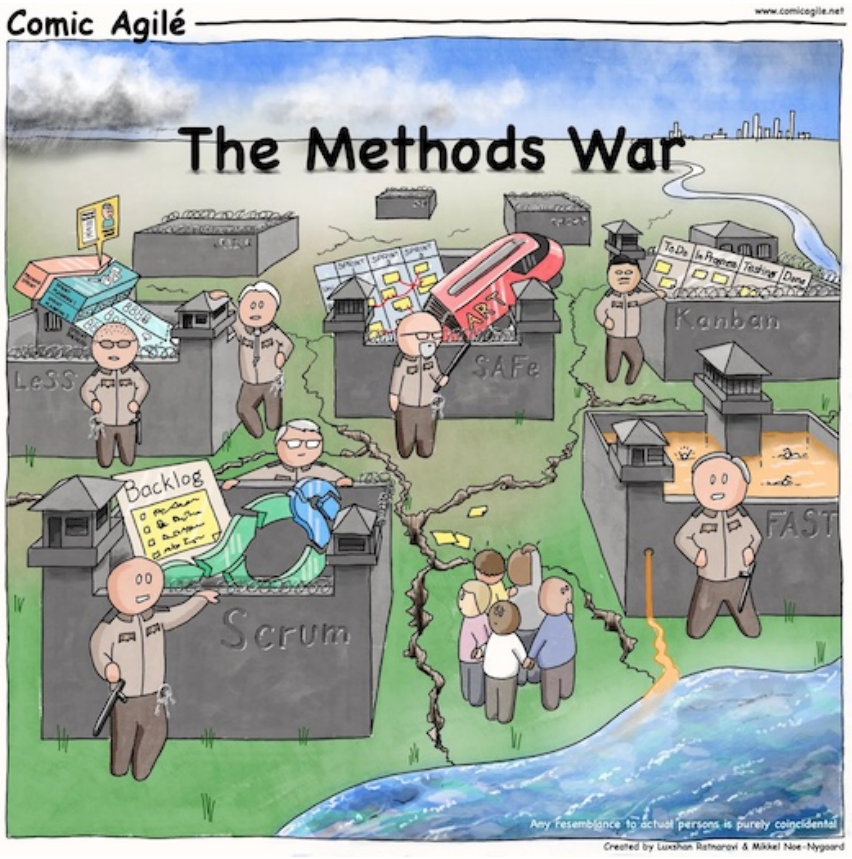

Figura 1.2 A (nova) Guerra dos Métodos, segundo os cartunistas-agilistas dinamarqueses Luxshan Ratnaravi e Mikkel Noe-Nygaard do excelente Comic Agilé.

Fonte: Página oficial do Comic Agilé.
<https://www.comicagile.net/comic/themethodswar/>.

Desenvolver software é resolver problemas complexos. Requer diferentes perspectivas, atitudes e capacidades. É um esporte coletivo praticado por profissionais com as mais variadas habilidades. Esperar que um método apenas seja suficiente e capaz de resolver todos os problemas dessa natureza, em todos

os contextos, é simplesmente infantil. O jornalista e ensaísta americano H. L. Mencken ilustra essa situação com uma frase muito bem humorada:

> *"Para todo problema complexo existe sempre uma solução simples, elegante e completamente errada."*

"Nossa maneira de desenvolver software…" E por que não? Que eu saiba, ninguém vai me processar por juntar o que há de melhor no XP, Scrum, Kanban, Management 3.0, Team Topologies, Flight Levels etc. e implementar uma solução que funcione para a minha organização em um determinado contexto. Aliás, essa é a palavra-chave: contexto. Na minha visão de Agilidade, não há espaço para dogmas, porque não existe uma solução, mas infinitas, e cada uma delas apropriada a um contexto específico.

O filósofo George Santayana nos alerta sobre o perigo de negligenciarmos o passado. O passado nos dá contexto, e contexto é tudo.

Como esse Livro está Organizado

Este livro foi organizado de forma cronológica, pois o objetivo principal é resgatar a história da Agilidade. Assim, no geral, cada capítulo explora um período subsequente ao capítulo anterior.

No primeiro capítulo, **"'Engenharia de Software' é um Oximoro?"**, exploro as primeiras tentativas de transformar o desenvolvimento de software em uma disciplina de engenharia, incluindo o período das grandes metodologias monolíticas estruturadas.

Em seguida, o capítulo **"Os Pioneiros"** (o mais longo do livro), foi dividido em duas seções:

- Na primeira seção, **"Os Primeiros Sinais de Luz no Fim do Túnel"**, eu resgato duas abordagens que já apontavam para um futuro diferente na atividade de desenvolvimento de software: RAD e Evo.
- Na segunda seção, **"Metodologias 'Leves' (Desculpe, Alistair Cockburn…)"**, relembro cada uma das abordagens que estavam representadas pelos seus principais criadores e praticantes, na reunião que deu origem ao Manifesto para o Desenvolvimento Ágil de Software.

Ao chegarmos nesse ponto do livro, estabelecemos o contexto que nos trouxe ao momento crucial que iria mudar os rumos do desenvolvimento de software e da gestão organizacional no século XXI: a reunião de 17 profissionais do software, em Utah, nos EUA, em fevereiro de 2001. Chamei este capítulo de **"O Que Temos Em Comum?"**, afinal esta era a motivação para aquele encontro.

Em **"Descobrindo Melhores Maneiras"**, examino algumas abordagens que vieram à luz após a publicação do Manifesto e contribuíram para a evolução do movimento. Longe de ser uma

lista exaustiva, é simplesmente um apanhado de modelos e ideias que considero as mais influentes e importantes para o atual estágio de evolução da Agilidade.

Finalmente, temos a conclusão com o capítulo **"Matando um Adjetivo"**, que discorre sobre o momento atual do movimento Ágil: os questionamentos, a confusão, as tentativas de assassinato (!), e as iniciativas para o resgate dos valores e princípios originais e essenciais do movimento.

Muitos se incomodam com o uso do termo "Agilidade", assim, com "A" maiúsculo. Escolhi essa grafia para representar o movimento iniciado a partir da publicação do Manifesto para o Desenvolvimento Ágil de Software, e tudo que estiver relacionado a ele. Sempre que leio agilidade com "a" minúsculo, penso no substantivo da língua portuguesa que significa a qualidade de ágil, sinônimo de presteza e vivacidade (conforme a definição no dicionário Priberam[3]), ou seja, algo muito mais genérico e pode causar confusão. Então, ainda que incomode alguns, continuarei usando Agilidade com "A" maiúsculo sempre que isso contribuir para dar total clareza de que estou me referindo ao movimento Ágil. (Nunca conseguiremos agradar a todos, como lembra a fábula do velho, o garoto e o burro — http://www.maisbelashistoriasbudistas.com/velho.htm.)

3. Definição de "agilidade" segundo o dicionário Priberam online: https://dicionario.priberam.org/agilidade

Espero assim contribuir na disseminação da Agilidade entre os lusofalantes, trazendo de volta suas origens, e algumas de suas evoluções, com a esperança de que cada vez mais o local de trabalho seja mais humano, centrado essencialmente nos indivíduos e suas interações, focado nas reais necessidades dos nossos clientes, e pautado pela excelência técnica. Em nosso mundo, cada vez mais turbulento, precisamos estar prontos para nos adaptarmos rapidamente às mudanças exigidas, e assim oferecermos as soluções certas, construídas da maneira correta,

para nossos clientes e usuários. Essa conquista requer muita cooperação e colaboração, entre todos os envolvidos: profissionais da tecnologia, camadas de liderança em nossas organizações, e principalmente, nossos clientes e usuários.

2. “Engenharia de Software” é um Oximoro?

“O desenvolvimento de software nunca será reduzido a um processo simples, mecânico de ‘linha de montagem’. A criatividade, os princípios da engenharia e a mudança evolucionária são necessários para a criação de um grande sistema satisfatório.”

Bjarne Stroustrup, cientista da computação e criador da linguagem C++*

* Dr. Stroutstrup é professor de ciência da computação na Columbia University, nos EUA, e criador da linguagem C++. A frase faz parte de uma entrevista publicada no livro “Masterminds of Programming: Conversations with the Creators of Major Programming Languages (Theory in Practice)”, de Federico Biancuzzi.

Oximoro

nome masculino

[Retórica] Combinação engenhosa de palavras cujo sentido literal é contraditório ou incongruente (ex.: bondade cruel é um oximoro). = OXÍMORO, OXÍMORON, OXIMÓRON

“oximoro”, in Dicionário Priberam da Língua Portuguesa [em linha], 2008-2024, https://dicionario.priberam.org/oximoro.

Fiat Lux! Nasce o Cérebro Eletrônico!

Ainda bem que havia luz, pois o programador solitário passou muitas madrugadas codificando a solução para aquele problema complicado que ele precisava resolver. No início, o trabalho de desenvolver software, ou como era chamado, então, programar, era uma atividade solitária, pois os programas eram muito mais discretos e limitados, quando comparados aos sistemas complexos e gigantes que temos hoje. No momento em que as necessidades se tornaram mais complexas, graças aos avanços vertiginosos da tecnologia do hardware, a atividade de desenvolver software foi ficando cada vez mais complicada. De verdade, complexa.

O hardware foi barateando, diminuindo de tamanho e aumentando em capacidade e velocidade. Consequentemente, essa evolução das máquinas requeria uma evolução daquilo que dava vida àquelas máquinas: software. Hardware sem software é só um pedaço de lata inútil. Sucata. Software é o que dá vida ao hardware.

Houve um tempo em que os computadores eram chamados de "cérebros eletrônicos" (isso foi lá na década de 1960, quando nasci). Em 1969, o grande Gilberto Gil até gravou uma música chamada "Cérebro Eletrônico" (sempre antenado ao *zeitgeist*):

O cérebro eletrônico faz tudo
Faz quase tudo
Faz quase tudo
Mas ele é mudo

O cérebro eletrônico comanda
Manda e desmanda
Ele é quem manda
Mas ele não anda

Só eu posso pensar
Se Deus existe
Só eu
Só eu posso chorar

Quando estou triste
Só eu

Eu cá com meus botões
De carne e osso. Hum hum
Eu falo e ouço. Hum hum
Eu penso e posso

Eu posso decidir
Se vivo ou morro
Por que
Porque sou vivo
Vivo pra cachorro
E sei

Que cérebro eletrônico
Nenhum me dá socorro
Em meu caminho inevitável
Para a morte

Porque eu sou vivo
Sou muito vivo
E sei

Que a morte é nosso
Impulso primitivo
E sei

Que cérebro eletrônico
Nenhum me dá socorro
Com seus botões de ferro
E seus olhos de vidro

Figura 2.1 A chegada do primeiro "cérebro eletrônico" no BNN em Recife, em 1965.

Fonte: Jornal Digital - Quais foram os primeiros computadores em Pernambuco. <https://jornaldigital.recife.br/2023/03/07/quais-foram-os-primeiros-computadores-em-pernambuco/>.

Se o computador era o cérebro eletrônico, então o software representava o conjunto das sinapses que faziam esse cérebro "pensar". E assim continua sendo sessenta e tantos anos depois. Usando outra analogia: o hardware é a marionete, e o software é o marionetista, aquele que dá vida ao objeto inanimado.

Logo começamos a falar em "sistemas" e não mais em "programas", pois os problemas complexos pediam soluções à altura. Sistemas são compostos por várias partes integradas, dado que um programa só não consegue resolver problemas grandes e complexos. O sistema então é constituído de vários programas integrados entre si e com as várias peças de hardware.

Quanto mais o hardware evoluía, mais complexo ficava o software. O computador primitivo deu lugar a um dispositivo sofisticado que permitia cada vez mais a interação com seu principal beneficiário:

o ser humano. Teclados, monitores, impressoras, mouses, mesas digitalizadoras, plotters, pediam software cada vez mais complexo. Imagine agora que temos tablets, smartphones, dispositivos de realidade virtual, internet das coisas, software embarcado em todos os lugares.

As soluções nunca são iguais. “Desenvolver software é desenvolver um novo produto”, afirma Craig Larman[1].

1. “Sofware Is New Product Development”, do livro “Agile and Iterative Development: A Manager’s Guide” [Lar04].

Para desenvolver sistemas tão complexos, era preciso bem mais do que um programador solitário. Desenvolver software se torna uma atividade coletiva. Agora precisamos de equipes de desenvolvimento que precisam se comunicar e se organizar muito bem para que seus trabalhos resultem em uma solução coesa, consistente: um verdadeiro sistema. Quando o programa foi substituído pelo sistema, tudo isso ainda era muito novo, e passamos naturalmente a nos inspirar em algumas ideias de outros paradigmas bem sedimentados, entre eles, a manufatura e a construção.

Em 1970, Dr. Winston Walker Royce[2] escreveu um importante artigo descrevendo sua experiência com o desenvolvimento e gestão de grandes sistemas de software, principalmente na indústria aeroespacial. O processo que ele descreve se assemelha muito a uma cascata, pois a informação flui em apenas uma direção: de cima para baixo, passando de uma fase de desenvolvimento para outra, como as águas de uma cascata que fluem morro abaixo, sem possibilidade de retorno. Ressalto que o Dr. Royce não inventou o método Cascata (*Waterfall*, em inglês), mas apenas descreveu um processo que já era bastante popular na época.

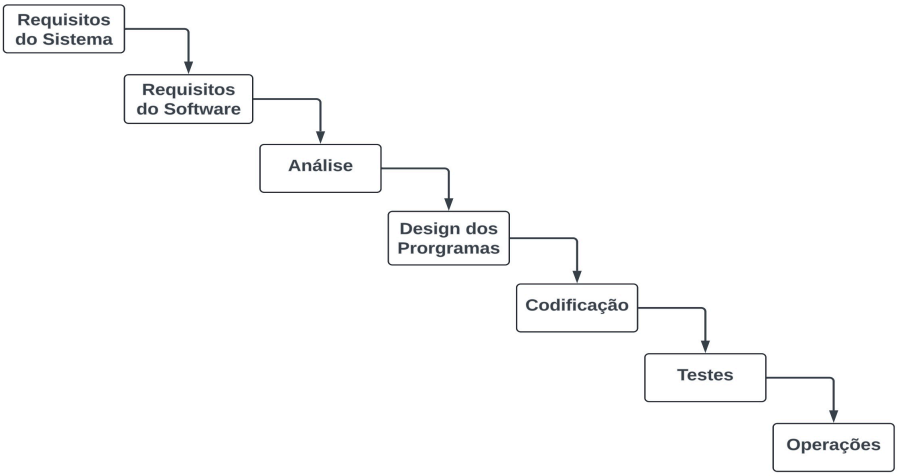

Figura 2.2 O exemplo do processo descrito no artigo do Dr. Royce em 1970.

Fonte: Figura adaptada pelo autor do original no artigo de Dr. Royce.

2. Link para o artigo original do Dr. Royce: https://www.praxisframework.org/files/royce1970.pdf.

O mais interessante (e trágico…) é que aparentemente ninguém deu muita importância ao que ele escreveu no restante do artigo; era como se a grande maioria só tivesse visto o bendito diagrama, e ignorado o resto. Entretanto, logo abaixo do diagrama, ele já alertava:

> *"Acredito neste conceito, mas a implementação descrita acima é arriscada e convida ao fracasso."*

Não é curioso? O método Cascata, em sua forma sequencial e linear, se torna então o grande paradigma de desenvolvimento de software, que perdurará por mais de duas décadas. A grande maioria das metodologias codificadas nas décadas de 1980 e 1990 reproduzia variações desse paradigma. E os fracassos foram se acumulando, como havia previsto o Dr. Royce.

O artigo em si é curto, tem apenas 11 páginas, e o diagrama reproduzido acima aparece logo na segunda página. O restante do artigo trata de formas adicionais ao modelo que poderiam eliminar os riscos previstos. Dr. Royce fala inclusive de iterações e protótipos. Mas aparentemente ninguém lhe deu ouvidos, e ele ficou conhecido erroneamente como o "pai do Cascata". Barry Bohem observou esse erro histórico em 1987[3].

3. https://en.wikipedia.org/wiki/Winston_W._Royce.

Todo o trabalho proposto pelos grandes metodologistas das décadas de 1980 e 1990, na procura de uma disciplina mais formal para o desenvolvimento de software, abriu caminho para novas formas de pensar e enxergar essa atividade tão essencial. O termo "engenharia de software" ganhou muita relevância nesse período (apesar de ter sido criado em 1968 em uma conferência da OTAN) — a crescente importância do software exigia uma nova forma de desenvolver.

Desenvolver software tinha que deixar de ser um esporte individual para se tornar um esporte coletivo. Sem regras claras, sem jogadores profissionais, sem tática, não temos futebol profissional de qualidade, mas "rachão" na praia! Quando o software começou a ser utilizado em instituições financeiras, equipamentos médicos, aeronaves e na indústria bélica, ele passa a ser um fator de risco com alto potencial de destruição. Neste cenário, surgiram as grandes metodologias de desenvolvimento de software do século XX, com a promessa de profissionalizar essa atividade com o rigor exigido.

Software é um Bicho Diferente

Meu primeiro contato com a programação de computadores foi no curso de engenharia civil na Escola Politécnica da Universidade de São Paulo. Logo no primeiro semestre aprendemos a programar em FORTRAN, e nossos programas eram escritos em formulários de papel, linha a linha, e enviados a um pool de digitadores. Esses utilizavam uma máquina que perfurava cartões de papelão (*punch cards*) — cada cartão representando uma das linhas de código, como preenchidas nos formulários. Depois de alguns dias recebíamos uma caixa com centenas de cartões perfurados, e estávamos prontos para testar nossos programas. Os furos seguem um padrão para representar os caracteres da linha de código, como pode ser visto na figura 2.3.

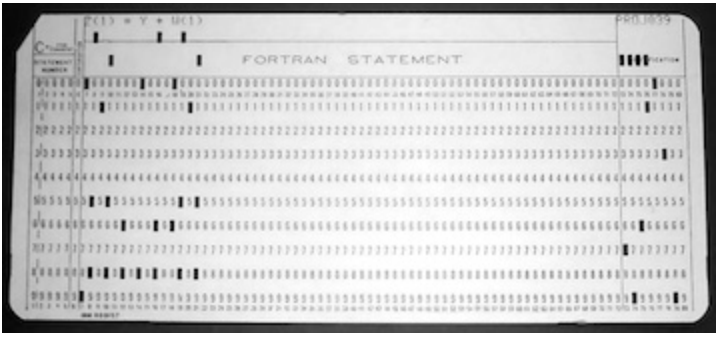

Figure 2.3 Cartão perfurado com uma linha de código FORTRAN — essa tecnologia foi inventada por Herman Hollerith no final do século XIX e adotada pela IBM em 1928.

Fonte: The Craft of Coding - Read your own punch cards <https://craftofcoding.wordpress.com/2017/01/28/read-your-own-punch-cards/>

Programação no Período Jurássico

No topo do cartão está a linha de código (algo como "Z(1) = Y + W(1)"), e no canto superior direito, temos a numeração do cartão (no exemplo, "PROJ039"). Lindo, não é? Só que muitas vezes, as máquinas perfuradoras usadas na universidade onde estudei não tinham tinta para imprimir nem a linha de código, nem o número do cartão… Já consegue imaginar alguns cenários de pesadelo?

Sem a linha impressa, era preciso consultar o formulário original com todas as linhas do programa como referência, mas isso só funcionava quando o aluno se lembrava de numerar os cartões manualmente, assim que eram recebidos.

Sempre tinha um infeliz que, estabanado, derrubava sua caixa de cartões e embaralhava todos eles, antes de ter a chance de numerar cada um. Era uma cena triste ver um colega chorando como criança, observando centenas de cartões espalhados pelo chão. Ah, os bons tempos de universidade…

Gosto da lenda que segue: — Era uma vez uma equipe de software que trabalhava em um sistema para controlar a aviônica[4] de um caça da força aérea americana. Peso é um fator essencial no projeto de qualquer aeronave, como se pode imaginar, e para lidar com isso o projeto contava com um "engenheiro de pesos". Um dia, esse engenheiro perguntou ao líder da equipe de software: "Quanto pesa o seu software?" E a resposta veio rapidamente: "Não pesa nada!" O engenheiro, inconformado, retrucou: "Já gastamos 15 milhões de dólares e não pesa nada?"

4. Aviônica é o nome que se dá para os sistemas de navegação e comunicação, piloto automático, controle de voo etc. de uma aeronave.

Alguns dias depois, o "engenheiro de pesos" retornou com uma caixa cheia de cartões perfurados contendo o software produzido pela equipe. "Aqui estão os cartões que juntos somam o peso do seu software, então só preciso pesar essa caixa, certo?" O líder da equipe de software então respondeu, um pouco sem jeito: "Mais ou menos, pois, na verdade, vamos ter que pesar os furos nos cartões." A lenda não conta se ele foi agredido, verbal ou fisicamente, pelo "engenheiro de pesos". Às vezes, a verdade dói.

Grady Booch[5] sintetizou muito bem a história acima:

> *"Um cartão perfurado da IBM pesa aproximadamente 2,54 gramas… mas, e o software em si? Ah, esse não pesa nada. Pois o software está nos furos e não nos cartões."*

5. https://twitter.com/Grady_Booch/status/1001548460124880897.

Essa característica abstrata, imponderável do software, é que faz com que ele seja um bicho diferente. Bem diferente do hardware. Ao perguntar aos meus alunos a diferença entre hardware e software, eu invariavelmente ouvia: "Hardware é aquilo que a gente chuta, e software é aquilo que a gente xinga, professor!"

Ao criar um novo modelo ou tipo de hardware, a equipe de produto com certeza estará focada na criatividade, no design e em toda a inovação envolvida na criação deste novo produto (o mesmo é verdade para carros, roupas, sapatos, brinquedos etc.) Mas uma vez que o produto tenha alcançado sua maturidade, ele entra no processo repetitivo de produção: cada um igualzinho ao outro. Variância é um dos demônios da manufatura, e para isso temos metodologias como o Six Sigma. Software, não! Sempre ouvi com estranheza alguém se referir a uma equipe de desenvolvedores que trabalham em uma "fábrica de software". Quem é o presidente desta fábrica? Frederick Taylor?[6]

6. Frederick Winslow Taylor foi um engenheiro mecânico americano, muito conhecido pelos seus métodos para melhorar a eficiência de indústrias. Considerado o pai da Administração Científica. (https://en.wikipedia.org/wiki/Frederick_Winslow_Taylor)

Desenvolver software é desenvolver um novo produto, lembrando mais uma vez o que nos disse Craig Larman. E por isso, o uso de paradigmas industriais ou da construção civil traz restrições gravíssimas ao processo de desenvolvimento. De fato, variância é uma característica marcante quando se desenvolve novos produtos por promover adaptação, inovação e evolução.

Em 1990, Peter DeGrace e Leslie Hulet Stal publicam o livro "Wicked Problems, Righteous Solutions" [DS90] e analisam as várias abordagens metodológicas disponíveis na época. É um tratado bastante abrangente, e traz um capítulo inteiro dedicado ao modelo Cascata, seguido por outro capítulo discutindo os problemas deste mesmo modelo. Os autores também exploram alternativas como os modelos incrementais, prototipagem, e até mencionam um tal de Scrum…

Logo na introdução, os autores trazem a seguinte frase:

> *"O campo da engenharia de software luta pela sua existência. É mais um ofício do que uma disciplina de engenharia, pois não se baseia em ciência."*

Um ofício e uma arte. Mas será que é também uma disciplina de engenharia? Susan Lammers tem algumas frases ótimas a respeito disso no seu livro "Programmers at Work" [Lam86]:

> *"… não sei que tipo de verdade a ciência de computação está tentando aprender." [p.55]*
>
> *"Considero (programação) como arte." [p.201]*
>
> *"… chamo (ciência da computação) um ofício, pois certamente ainda não é uma ciência." [p.216]*

Alistair Cockburn é um dos meus autores preferidos quando o assunto é desenvolvimento de software. Ele é doutor em metodologias de desenvolvimento e tem se debruçado sobre o

assunto por toda a sua carreira; também é um dos signatários originais do Manifesto, como veremos mais adiante. Em seu livro "Agile Software Development: The Cooperative Game" [Coc06], ele reconhece que a natureza do software é múltipla, e traz uma perspectiva bem abrangente:

> *"Embora programar seja uma atividade solitária, lógica e baseada em inspiração, é também uma atividade de engenharia em grupo. É um paradoxo porque desenvolver software é ao mesmo tempo:*
>
> - *Matemática*
> - *Engenharia*
> - *Ofício*
> - *Arte*
>
> *"É uma atividade cognitiva e expressiva realizada por pessoas que pensam e se comunicam, trabalhando com restrições orçamentárias e técnicas, condicionadas às suas culturas, e sensível às pessoas especificamente envolvidas."*

Então, engenharia de software não é um oximoro, Alistair? Ele continua:

> *"Na minha experiência, as pessoas confundem as atividades de **fazer** engenharia com o **resultado** da engenharia. O **resultado** de fazer o trabalho de engenharia é a fábrica, que produz sob a supervisão de pessoas, observando cuidadosamente variações na quantidade e qualidade dos itens sendo produzidos.*
>
> *"O ato de fazer o trabalho de engenharia consiste em um processo criativo, mal definido, que o engenheiro industrial realiza para inventar o design da fábrica. É um jogo cooperativo de invenção e comunicação, como também é o desenvolvimento de software, onde pessoas com experiências diferentes se juntam para desenhar uma solução viável."*

O sentido original da palavra "engenharia" vem de "engenhar" que significa idear, inventar. Neste sentido, desenvolver software é sim uma atividade de engenharia. Então, é perfeitamente cabível falar em engenharia de software, quando adotamos a perspectiva de engenharia como invenção, criação.

As tentativas de trazer mais rigor e formalidade à atividade de desenvolver software, nas décadas entre 1970 e 1990, foram fundamentais e precisam ser valorizadas e reconhecidas por todos nós, profissionais do software. Ocorre que cada contexto pede uma abordagem apropriada quando se trata de desenvolvê-lo. Um dos principais erros de muitas metodologias monolíticas (assim chamadas por serem muito grandes) era imaginar que suas propostas configuravam um método unificado para resolver todos os problemas em todos os contextos. Algumas pessoas ainda insistem em procurar, e acreditar, na pedra filosofal do desenvolvimento de software. Contudo, o sucesso no desenvolvimento de software é pautado por duas ideias centrais: contexto e adaptação.