Licenciatura em Gestão de Sistemas de Computação

A importância da comunicação, no contexto do desenvolvimento de Software

Projeto Final de Licenciatura

Elaborado por Pedro Martins

Aluno nº 20101426

Orientador: Prof. Joaquim Canhoto

Barcarena

Fevereiro de 2014

Universidade Atlântica

Licenciatura em Gestão de Sistemas de Computação

A importância da comunicação, no contexto do desenvolvimento de Software

Projeto Final de Licenciatura

Elaborado por Pedro Martins

Aluno nº 20101426

Orientador: Prof. Joaquim Canhoto

Barcarena

Fevereiro de 2014

O autor é o único responsável pelas ideias expressas neste relatório

Agradecimentos

A todas as pessoas que me querem, ou hão de querer bem, durante toda a vida. Elas sabem ou saberão quem são, tenha eu sabedoria e hombridade para lhes reconhecer o mérito.

Agradeço acima de tudo, a minha família mais chegada, pelo amor incondicional que me dão, meus pais e irmão, a mãe do meu filho, e este ultimo que tem hoje 4 anos. Esse piolho lindo que nasceu com a minha vontade tardia de ser pai e ser estudante. Peço-lhes eterno perdão pela ausência da minha presença, derivado aos estudos. Espero agora poder compensa-los a dobrar. Agradeço-lhes pela paciência e apoio dado ao longo desta jornada longa e dura. Sendo que ainda a valorizo mais (a jornada) por ter sido realmente dura, porque de coisas fáceis está o mundo cheio. Fecha se um ciclo, com este trabalho.

Espero que ao fechar-se uma janela, possa eventualmente abrir-se uma porta.

Agradeço a todos os que me proporcionaram as boas e más experiencias vividas ao longo de toda a licenciatura, temos de ter a capacidade de tirar de ambas, bons ensinamentos. Pois não aprendemos apenas com o sucesso e coisas boas, mas acima de tudo, e até talvez muito mais, com os nossos próprios fracassos e erros.

Quero agradecer ao Professor Joaquim Canhoto, não só pelo seu acompanhamento dado neste trabalho, mas acima de tudo pela pessoa que é. Com muita pena minha não privei tanto com ele, como gostava de ter privado, pois apenas tive o prazer de o fazer no ultimo ano da licenciatura.

Quero agradecer à Dra. Cristina Soares, pelo contributo dado, para este trabalho.

Quero Agradecer, a todos os outros professores e colegas, que me acompanharam ao longo destes 3 anos, não quero fazer distinções para não se chatearem, os mais importantes para mim, sabem quem são!

“Mudam-se os tempos, mudam-se as vontades,

Muda-se o ser, muda-se a confiança;

Todo o mundo é composto de mudança,

Tomando sempre novas qualidades.

Continuamente vemos novidades,

Diferentes em tudo da esperança;

Do mal ficam as mágoas na lembrança,

E do bem, se algum houve, as saudades.”

Excerto de Soneto de Luís de Camões (1524?-1580)

Resumo

A importância da comunicação, no contexto do desenvolvimento de Software

Em média apenas 39% dos projetos de desenvolvimento de *Software* são concluídos, entregues no prazo, dentro do orçamento, com as funções e os recursos necessários. No entanto, nos últimos 8 anos temos assistido a uma melhoria de 10% neste valor. Isto tem tudo a ver com a adoção nos últimos anos de valores e princípios "ágeis". Devemos estar a melhorar de alguma forma o que estamos a fazer, no entanto ainda há um longo caminho a percorrer, porque uma grande percentagem das pessoas e equipas envolvidas, ainda continua a fracassar. Sabemos que a participação dos utilizadores desempenha um papel importante na obtenção do sucesso.

Sabemos também que uma grande parte das falhas ocorre não por causa de uma má escolha de implementação do modelo ou a falta de "capacidades técnicas" das pessoas envolvidas. A maior causa de falhas no desenvolvimento de *Software* tem a ver com o levantamento e análise de requisitos incorretos ou incompletos. Isto deve-se principalmente à má comunicação (aptidão) e problemas relacionados com a perceção entre todas as pessoas envolvidas. Não é que a equipa de desenvolvimento não saiba como fazer as coisas corretamente, ela só não está a fazer o que precisa de ser feito. São problemas de pessoas, da sua falta de cooperação e comunicação, que ao longo dos anos teimosamente tentamos resolver com soluções técnicas, com muito pouco sucesso. Após a análise que eu fiz, cheguei à conclusão de que deveríamos tentar estudar e criar soluções que tenham abordagens mais sociológicas ou psicológicas.

Palavras-chave: processo de Desenvolvimento Software (modelos), comunicação, pessoas, perceção, motivação.

Abstract

The importance of communication, in the context of Software development

On average only 39% of software projects are completed, delivered on time, on budget, with required features and functions. Never the less, we have seen a progress of 10% over the last 8 years. This has everything to do with the adoption of "Agile values and practices, over the last years. We must be doing something better, but there is still a long way to go, because a great percentage of the people and teams involved, still keep on failing. We know the participation of users, in software development, plays a big role in achieving success.

We also know that most of the failures occur, not because of a bad choice of model, it´s implementation or the lack of "technical capabilities of the people involved. The biggest single cause of failure in software development is bad or incomplete requirements gathering and analysis. This is mainly due to bad communication (skills) and problems related to perception between all the people involved. It not that the development team doesn´t know how to do things correctly, they are just not doing what needs to be done.

It´s a people problem, lack of collaboration and communication, that over the years we have stubbornly tried to solve with technical solutions, with almost no success. After the analysis I made, I concluded that we should be trying to study and create solutions that have more of a psychological or sociological approach.

Keywords: Software Development process (models), communication, people, perception, motivation.

Índice

Índices de figuras

Índices de ilustrações

1. Introdução

Com este capítulo introdutório, pretende-se antes de mais fazer uma contextualização histórica, de forma a perceber melhor onde se enquadra a temática deste trabalho final de licenciatura. Posteriormente é feita uma análise do problema e uma proposta de desenvolvimento do trabalho. Finalmente é explicada toda a estrutura do trabalho.

1.1. Contextualização histórica[1]

O ser humano desde os seus primórdios sentiu a necessidade de representar de alguma forma os objetos do seu quotidiano, recorrendo para isso, ao desenho em diferentes superfícies e materiais.(Anon 2014) Esta atividade está intrinsecamente ligada à necessidade que o ser humano sempre teve:

- Em comunicar
- Deixar registadas a suas vivências
- Partilhar o seu conhecimento

Ilustração 1 - Felinos Predadores. Chauvet, França c. 30,000 AC. [2]

Uma das primeiras representações de arte rupestre.

[1] Http://www.historyofinformation.com/

[2] Http://www.ancient-wisdom.co.uk/caveart.htm

Inicialmente, o Homem limitou-se a usar superfícies e materiais que a natureza lhe oferecia, tais como:

- Paredes rochosas
- Pedras
- Ossos e Peles de animais
- Folhas de certas plantas

Através do desenvolvimento intelectual progressivo do ser humano, e com o recurso a utensílios e materiais criados ou transformados por ele, foi desenvolvendo suportes cada vez mais adequados a representações gráficas mais elaboradas.

Com esta finalidade, a história regista o uso de:

- De barro cozido
- Tecidos de fibras diversas
- Papiros e pergaminhos
- Finalmente, papel.

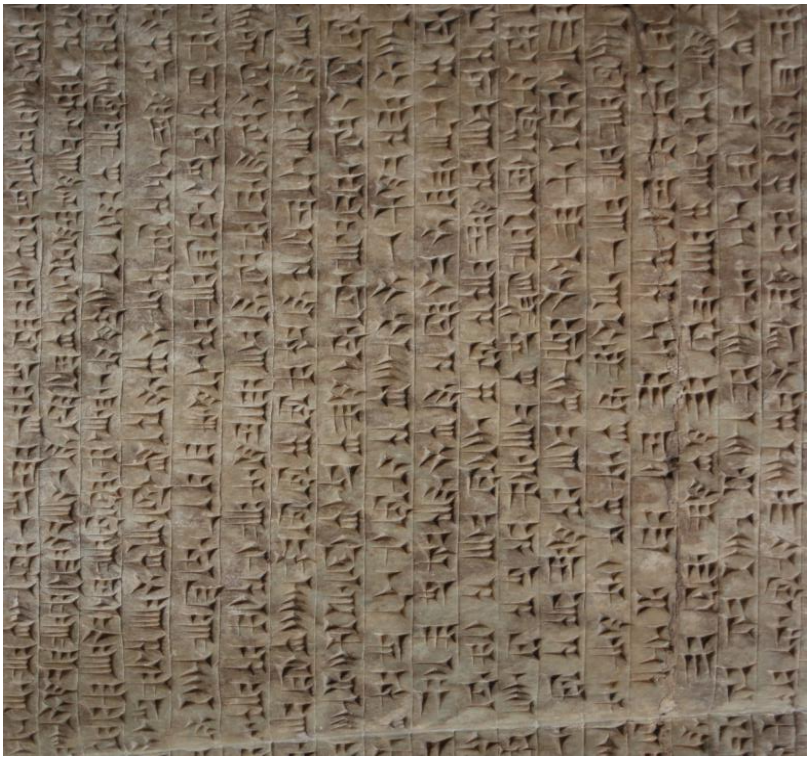


Ilustração 2 - Escrita cuneiforme: tabuleta de barro, Mesopotâmia, 3100-2800 AC[3]

[3] Http://www.ancient.eu.com/image/93/

Todas estas representações gráficas foram se tornando cada vez mais complexas, passando desse modo a significar ideias. Este desenvolvimento culmina com o aparecimento da escrita cuneiforme, criada pelos Sumérios, na Mesopotâmia, por volta de 5000 anos antes de cristo. Esta surge inicialmente com a necessidade de manter registos contabilísticos, nomeadamente de produtos comercializados, impostos arrecadados, nº de funcionários do Estado etc. (Guisepi 1999)(Mark 2011)

A escrita foi fundamental para o desenvolvimento do ser humano, sendo a principal responsável por grande parte das inovações que lhe são posteriores. O que torna a escrita tão especial é o facto de esta nos permitir ter acesso ao registo de todo o conhecimento adquirido e guardado no passado.

Deixamos de estar limitados à transmissão do conhecimento pela via oral e todas as desvantagens que envolvem perpetuar o conhecimento desta forma. "Basta imaginar a potencial perda de grande parte do conhecimento acumulado ao longo de seculos, que uma grande epidemia ou cataclismo poderiam causar, numa sociedade onde todo o conhecimento seja transmitido pela via oral". (Kaviratna 1971)

Durante cerca de 7000 anos, e até algumas décadas atrás, todo o tipo de dados foi sendo armazenado com o recurso à escrita, com o recurso a vários tipos de suporte, com um aumento da prevalência da impressão em papel nos últimos 500 anos. (bellis n.d.) Esta forma de registo de dados foi crucial para a perpetuação e disseminação do conhecimento.

No entanto também a impressão em papel teve (e tem) algumas limitações, por exemplo, torna-se difícil ou pelo menos pouco prático a consulta, cruzamento e análise de dados neles contidos (em tempo oportuno), pelo facto de estes poderem se encontrar dispersos em vários documentos, livros, diferentes locais etc.).

Existem no entanto vários autores que criticam a massificação da impressão em papel, levada a cabo, a partir da invenção de Gutenberg. McLuhan por exemplo chamou-lhe a 'tecnologia do individualismo', por defender que esta potencia o afastamento entre as pessoas. (Evans 1999)

A leitura é sem dúvida uma atividade maioritariamente solitária, que deve de ser feita em silêncio. Permitindo deste modo que as pessoas desenvolvessem um sentimento cada vez maior de privacidade pessoal, tão patente nas sociedades atuais. O aumento da literacia teve um papel preponderante na uniformização de dialetos regionais, o que ajudou no crescimento consciências nacionalistas, com tudo o que de negativo dai conseguiu vingar (xenofobia, racismo). (Evans 1999)

A leitura limita de certa forma a capacidade de conseguirmos interpretar o que o autor na realidade queria dizer, não conseguimos por exemplo, fazer uma pergunta de esclarecimento ao autor (pelo menos em tempo real). É por vezes necessário, fazer uma contextualização à época em que o texto foi escrito, perceber quem era o autor no sentido de perceber as suas intenções ou o sentido do que este pretendia dizer, os quais podem não estar expressos no texto. (Richards n.d.)

Felizmente o suporte em papel tem vindo progressivamente a ser substituído por dados em suporte digital com a ajuda das "tecnologias da informação", termo que surge pela primeira vez, num artigo de 1958 do Harvard Business Review escrito por Harold J. Leavitt e Thomas L. Whisler.

O uso dos computadores veio permitir o armazenamento e tratamento de quantidades imensas de dados, estes permitem a automação de muitos processos repetitivos, reduzindo o tempo necessário para os executar, permitindo assim reduzir custos e maximizar recursos.

No entanto convém lembrar…

"A primeira regra de qualquer tecnologia utilizada num negócio é que a automação aplicada a uma operação eficiente ampliará a sua eficiência… A segunda é que a automação aplicada a uma operação ineficiente ampliará a sua ineficiência."

Segundo Bill Gates[4]

[4] Http://www.brainyquote.com/quotes/quotes/b/billgates104353.html#cWEaJcbc80iXBqlq.99

Foi de facto na primeira metade do seculo 20 que apareceram os primeiros computadores verdadeiramente programáveis. No entanto, e pelo menos numa fase inicial, o conhecimento necessário para programar num destes computadores era muito específico, devido à necessidade do uso de Linguagens de programação de baixo nível. Este facto limitou durante os primeiros tempos o seu uso quase exclusivamente aos seus criadores. No entanto, e com o passar dos anos, fomos assistindo a massificação do uso dos mesmos. Este fenómeno deve-se em grande parte à evolução das linguagens de programação. (Date 2000)

Esta evolução demonstra claramente uma procura que tem estado sempre presente nos nossos espíritos desde os primórdios (pelo menos de alguns), e que é transversal a todas as áreas do conhecimento. Existe uma busca de uma redução constante:

- Do esforço e tempo gasto com as tarefas de desenvolvimento
- Do custo associados a estas tarefas

Temos vindo a assistir nas últimas décadas, a um crescimento exponencial do uso das tecnologias de informação. Segundo Dates “isto deve-se a um contínuo crescimento dos níveis de abstração das linguagens de programação”, (Date 2000) estes permitem que hoje possamos usar um computador eficazmente sem ter um conhecimento efetivo do funcionamento interno do mesmo. O facto de hoje, não precisarmos de “falar” a língua dos processadores que estão nos nossos computadores para podermos efetivamente interagir com os mesmos, deve-se (entre outras coisas), à existência destes níveis de abstração (camadas).

As tecnologias da informação são hoje essenciais para que se consiga tratar todo o conhecimento e informação acumulado de uma forma eficiente e se possível em tempo real. E se estas, de início, eram consideradas apenas como uma ferramenta extra para resolver tarefas simples, com o tempo a sua importância foi crescendo, à medida que estas têm vindo a ser usadas para solucionar problemas cada vez mais complexos e abrangentes.

Pressman afirma que "vivemos num mundo em constantes e rápidas mudanças. Este ritmo de mudança a nível das funções de negócio e das tecnologias de informação que lhe dão suporte, exerce uma enorme pressão competitiva sobre todas as organizações" (Pressman 2009)

Existe assim hoje uma cada vez maior complexidade e consequente aumento da incerteza, inerentes a grande parte dos projetos de desenvolvimento de *Software*. Na medida em que os sistemas que suportam as TI se tornam também eles mais complexos. Estes exigem por isso uma cada vez maior capacidade de gestão de recursos, quase sempre escaços como tempo, dinheiro e pessoal qualificado.

Estes factos fizeram com se fossem tomando ao longo dos anos abordagens necessariamente diferentes e mais adaptativas ao desenvolvimento de *Software*. De facto, temos assistido à adoção gradual de novos modelos de desenvolvimento, que procuram "agilizar" o modo como se aborda um projeto, de forma a conseguir contornar dificuldades associadas ao aumento de complexidade, incerteza e ritmos de mudança.

Neste contexto, este trabalho procura fazer uma análise da importância da comunicação, no contexto do desenvolvimento de Software.

1.2. Problema

Existe o mito de que o que mais importa nas tecnologias de informação são os computadores e as tecnologias usadas, na realidade o que importa mesmo, são as pessoas. (Al Neimat 2005; Betts 2003) Porque são as pessoas que criam a tecnologia, então a tecnologia não resolve os problemas, quem os resolve são as pessoas.

Jeff de Luca, um dos autores do "Feature Driven Development", diz que a primeira lei das TI deverias ser que, "As TI são 80% psicologia e 20% tecnologia". De facto "a natureza de grande parte dos problemas do nosso trabalho…" relacionados com a gestão de projetos "…não é tanto tecnológica, mas sim sociológica". (DeMarco & Lister 1987)

Varios autores referem que as pessoas, a forma como se relacionam, estão organizadas (cultura corporativa) e são geridas, são factores muito mais importantes no sucesso de um projeto, do que quais queres ferramentas, técnicas ou processos usados. (Brooks Jr 1995; Andriole 2011). De facto, uma grande parte da gestão do desenvolvimento de *Software* está associada à identificação e redução de riscos, sendo que uma grande parte destes está na realidade relacionada com pessoas ou processos (em que estas estão envolvidas).

"Dito de forma simples, as TI não funcionam se estiverem rodeadas de pessoas incompetentes, processos estúpidos e uma cultura corporativa desastrada". (Andriole 2011)

Segundo Brooks "a parte mais difícil de todo o processo de desenvolvimento de um sistema de *Software* é decidir precisamente o que se vai construir. Nenhuma outra parte do trabalho conceptual é tão difícil como a de estabelecer, de forma detalhada, os requisitos técnicos"…"quando feita de forma errada, nenhuma outra parte do processo é tão difícil de retificar posteriormente e tem tanto impacto negativo no sistema dai resultante". (Brooks Jr 1995)

De facto, para quem desenvolve, conseguir idealizar um produto ou sistema a partir de necessidades (requisitos) que nem sempre são claras, não é tarefa fácil. Conforme Wiegers e Beatty afirmam, "as pessoas (clientes) têm normalmente alguma dificuldade

em conseguir descrever o que querem, sem serem (primeiro) confrontadas com algo que lhes seja minimamente tangível…" (Wiegers & Beatty 2013)

Boehm usa a seguinte sigla IKIWISI, que provem do inglês:

"I'll Know It When I See It"

…para identificar o problema da falta de capacidade que o cliente tem em conseguir definir o que quer de uma forma clara, quando este afirma que "saberá o que quer quando o vir". (Boehm 2000) "O cliente precisa de algo que possa, de certa forma, servir de termo de comparação…" É-lhe certamente mais fácil descrever aquilo que não quer, do que aquilo que quer, "…criticar é sempre mais fácil do que criar". (Wiegers & Beatty 2013)

Mas, se para um cliente "uma imagem pode valer mais que mil palavras", Brooks defende de certa forma o contrário, pelo menos na perspetiva de quem está do lado oposto (a equipa de desenvolvimento), quando diz que "se me mostrares os teus diagramas e ocultares as tuas tabelas, eu continuarei perplexo. Se me mostrares as tuas tabelas, normalmente não precisarei de ver os diagramas, as informações neles contidos serão óbvias". (Brooks Jr 1995)

Talvez seja então mais logico fazermos a seguinte afirmação,

"Uma imagem pode ajudar a definir mais de mil palavras"

Para quem desenvolve, e mesmo após ter feito o levantamento de requisitos através dos métodos mais comuns, poderá persistir (quase sempre) algum nível de incerteza em relação a uma ou mais partes do projeto. Caso não se procure clarificar essas incertezas, e se avance para o desenvolvimento, poderá surguir algum desfasamento entre as expectativas criadas pelo cliente, e as criadas pela equipa de desenvolvimento. Passando a existir efetivamente um fosso entre essas expectativas, em resultado de diferenças de entendimento em relação ao que deve de ser o produto final. (Wiegers 1995)

Existe pois a necessidade de eliminar ou reduzir ao máximo o fosso que possa existir entre a visão que o cliente tem do produto, e o entendimento que a equipa de desenvolvimento possa ter do mesmo. Entre as várias ferramentas disponíveis para quem desenvolve, a prototipagem é uma das mais eficazes para ajudar a reduzir esse fosso. Através do uso de vários tipos de protótipos consegue-se transpor para a realidade "use cases", "user stories", etc. de forma a conseguir colmatar lacunas de entendimento que possam existir ao nível dos mesmos.

Mas qual a razão fundamental para termos de recorrer ao uso de ferramentas como a prototipagem? O problema no meu entender, tem a ver com o facto de a comunicação ser realmente complicada e revestida de varias camadas de abstração, que não permitem nem mesmo nas mensagens mais claras, uma perceção clara destas mesmas. Não sei se isso se deve apenas à complexidade da comunicação, aos meios usados nesta ou à própria complexidade e diversidade de quem recorre à mesma (pessoas).

Mas se temos, o conhecimento, dispomos de muitos modelos, metodologias e *Frameworks* que nos podem ajudar a resolver os problemas, qual a razão (ou conjunto delas) para tanto insucesso no desenvolvimento de *Software?*

O Standish Group analisa dados referentes ao sucesso de projetos de desenvolvimento de *Software* desde 1985, No seu relatório anual, denominado "Chaos Manifesto".

O seu mais recente relatório é de 2013 (relativo a dados de 2012). Neste, foram analisados dados referentes a cerca de 50000 projetos recolhidos em todo o mundo, dos quais, 60% são dos Estados Unidos, 25% são da Europa e 15% são de outros países.

Neste relatório o Standish Group define os 3 critérios seguintes para fazer a avaliação do sucesso dos projetos em análise: (Standish Group 2013)

- Se a entrega foi atempada
- Se foi cumprido o orçamento estabelecido
- Se as funcionalidades e recursos disponibilizados correspondem ao que estava definido para o projeto.

Chegando desta forma aos seguintes resultados:

- 18% dos projetos fracassaram completamente (cancelados antes da conclusão ou entregues mas nunca usados)
- 43% dos projetos foram considerados problemáticos porque não cumprem pelo menos um dos critérios apresentados acima.
- 39% dos projetos são considerados bem-sucedidos, por cumprirem com os 3 critérios.

Em qualquer outra indústria não seria certamente aceitável tão elevado grau de insucesso. Mesmo assim houve uma melhoria de 10% em termos de projetos bem-sucedidos no prazo de 8 anos, mantendo-se no entanto o valor de 18% de projetos fracassados.

Se de 100 casas por construir, apenas 39% fossem concluídas, 61% sofressem algum tipo de desabamento e 18% desabassem na totalidade antes de concluídas, seria sustentável tal situação? Não estamos a falar de casas, mas sim de *Software*, não tem talvez a mesma criticidade, porque não corremos o risco de existir danos físicos ou morte de pessoas. Mas em termos de perdas medidas por homem/hora, os valores obtidos não deixam de ter um caracter dramático.

Então mas qual é o problema (ou conjunto deles) para tanto fracasso?

Existe forma de melhorar o panorama atual?

Não prometo descobrir a solução, mas procurarei pelo menos até ao final deste trabalho conseguir, nem que seja apenas pessoalmente, ficar um pouco mais elucidado sobre toda esta problemática. O que para mim já seria certamente uma vitoria retumbante!

1.3. Proposta

No sentido de perceber qual a importância da comunicação no desenvolvimento de *Software*, convém quanto a mim, antes de mais responder as seguintes perguntas:

- O que é um processo de desenvolvimento de *Software* (seu ciclo de vida)?
    - Qual a razão da sua existência?
    - Quais as atividades no processo de desenvolvimento?
    - Porque se considera o desenvolvimento de *Software* um tarefa difícil?

- Quais os modelos disponíveis?
- Quais as diferenças entre as diversas abordagens feitas no desenvolvimento?
- Quais os fatores de sucesso para o desenvolvimento de *Software?*

A resposta a estas perguntas, e a análise de toda a informação recolhida, permitirá chegar a conclusões "minimamente" estruturadas, de forma a conseguir definir um caminho solido a seguir em trabalhos futuros.

1.4. Estrutura do trabalho

Este trabalho encontra-se organizado em 5 capítulos:

No primeiro capítulo é feito uma contextualização histórica, de forma a melhor perceber a problemática do tema abordado. Posteriormente é feita uma análise mais aprofundada do problema e definida uma proposta de desenvolvimento do trabalho (onde se enquadra também esta estrutura do trabalho).

No Segundo capítulo são apresentados os conceitos básicos e trabalhos relacionados para uma correta compreensão do problema da comunicação no contexto do desenvolvimento *de Software*, como seja:

- O processo de desenvolvimento de *Software*
- As várias abordagens
- Os vários tipos de modelos

O Terceiro capítulo aborda os fatores que influenciam o sucesso / insucesso do Desenvolvimento de Software.

No Quarto capítulo é feita revisão de toda a informação recolhida, esta é contextualizada em função da importância das pessoas e da comunicação entre estas, no desenvolvimento de *Software*.

No Quinto e final capítulo são feitas a considerações Finais e desenvolvidas as ideias para um trabalho futuro, no sentido de conseguir materializar os conhecimentos adquiridos, neste trabalho. Também são abordadas as soluções existentes no mercado e as melhorias que este documento pretende retratar, por forma a preencher algumas lacunas nos sistemas atuais.

2. Conceitos Básicos e Trabalhos Relacionados

Neste capítulo será feita uma revisão bibliográfica que aborda:

- A razão da existência de um modelo de desenvolvimento de *Software.*
- A definição do que são estes modelos.
- As atividades essenciais destes modelos.
- Desenvolver *Software* é difícil. Analise dos fatores principais que influenciam a esta perceção generalizada.
- Uma análise aos vários modelos existentes na literatura.
- As várias abordagens possíveis no desenvolvimento de *Software.*
- Os fatores que têm influência no sucesso do desenvolvimento de *Software.*

…No sentido de perceber a importância relativa da comunicação no contexto do desenvolvimento de *Software*. Procurando apresentar também, uma visão sobre os problemas envolvidos em todo o processo.

2.1. Processo de desenvolvimento de Software (modelos)

Deming[5] afirma que,

> **"Se não conseguimos descrever o que estamos a fazer, como um processo, então não sabemos o que estamos a fazer"**

(Segundo Pressman) Baetjer afirma acerca do processo de desenvolvimento de *Software*, o seguinte:

" Porque o *Software*, como todo o capital, é composto por conhecimento, o qual:

- Está inicialmente disperso, e algo incompleto.
- É tácito (Conhecimento normalmente adquirido ao longo da vida, pela experiência. É por isso por vezes difícil de ser formalizado ou explicado a outra pessoa, pelo facto de ser subjetivo e estar inerente a cada individuo).
- Está latente (é o conhecimento que cada individuo regista com a experiencia, mas que fica armazenado (adormecido) em cada um de nós, que pode em qualquer altura, por exemplo servir para a tomada de uma decisão).

O processo é um diálogo (aprofundar da aprendizagem e do entendimento), em que todo o conhecimento que tem de ser transformado em *Software* é reunido e incorporado dentro do desse mesmo *Software*. O processo proporciona uma interação contínua entre os utilizadores, equipa de desenvolvimento e o *Software*, que estão todos em constante evolução. Estamos perante um processo iterativo no qual o próprio *Software* serve de catalisador da comunicação, em que a cada novo ciclo se consegue extrair cada vez mais conhecimento útil das pessoas envolvidas." (Pressman 2010; Baetjer 1997)

De facto conforme Pressman afirma, "criar *Software* é um processo iterativo de aprendizagem social", o resultado, conforme Baetjer lhe chama de "*Software* capital," é

[5] Http://www.brainyquote.com/quotes/quotes/w/wedwardsd133510.html

o encarnar de conhecimento, colhido, destilado e organizado ao longo do desenrolar de todo o processo de desenvolvimento. (Pressman 2010; Baetjer 1997)

Os primeiros modelos desenvolvimento de *Software*, ou conceitos iniciais destes, surgem por volta do final da década de 50 do seculo passado, como forma de resposta à falha de vários projetos de desenvolvimento relevante. (Sommerville 1996)

Estes modelos são importante (Boehm 1988), por proporcionarem orientação quanto a:

- Que tarefas são necessárias executar e a sua duração (etapas)
- Quem as executa
- Qual a sua ordem (critérios de escolha / evolução)
- Forma como se transita de uma etapa para outra (critérios de conclusão / transição)

Através dessa definição, espera-se conseguir desenvolver de forma organizada esse mesmo *Software*. Fornecendo a todos os envolvidos, informações exatas e relevantes a cerca do projeto em desenvolvimento. A intenção é conseguir entregar o *Software* atempadamente e com a qualidade suficiente, de forma a conseguir satisfazer, tanto aqueles que patrocinam a sua criação, como aqueles que irão usá-lo (tanto quanto possível). (Pressman 2010)

Procurar-se desta forma, garantir o nível de qualidade pretendida, através do controlo destas 3 variáveis de:

- Funcionalidades (Âmbito)
- Prazos (Tempos de entrega)
- Custos

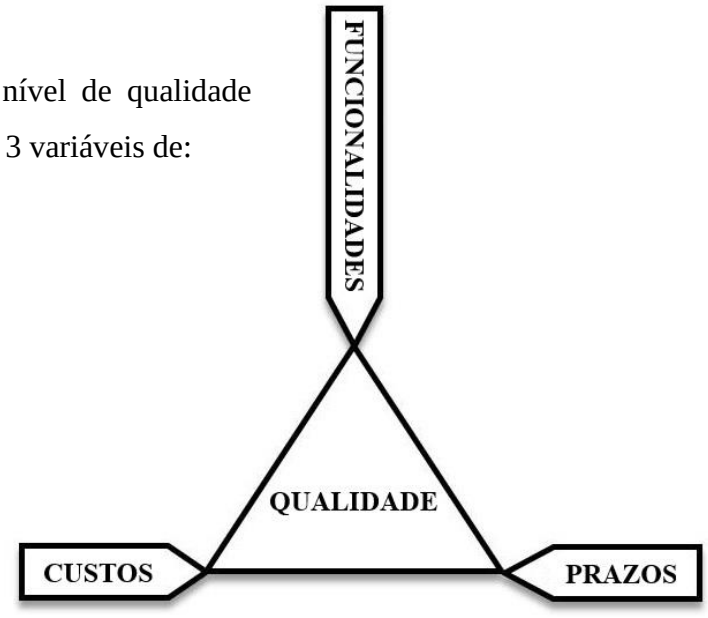


Figura 1 - Triangulo da gestão de projetos[6] (adaptado do original)

[6] Http://en.wikipedia.org/wiki/File:The_triad_constraints.jpg

Na gestão e desenvolvimento de Software, conseguir controlar estas 3 variáveis, é essencial para o sucesso de um projeto, no entanto isto pode ser mais difícil do que parece. Isto porque estamos perante um trilema, ou seja apenas conseguimos (normalmente) obter os níveis desejados em duas das três variáveis, à custa de uma 3 variável. (Atkinson 1999)

Provavelmente não será possível concluir um projeto que possa garantir algum nível de qualidade, se pretendermos que o mesmo seja executado de uma forma rápida e barata.

Perante a necessidade de conseguir manter os níveis de qualidade e o prazo de entrega (conclusão), podemos ter de aumentando por exemplo o número de pessoas envolvidas no projeto, o que pode até garantir prazo e qualidade do projeto, mas estes são alcançáveis em detrimento do aumento dos gastos.

No caso de o aumento de custos não ser viável ou desejável, teremos de abrir mão de um dos outros dois fatores:

- Ou se aumenta o prazo de entrega, com uma redução das pessoas envolvidas (manutenção de custos), mantendo o nível de qualidade
- Ou se reduz o nível de qualidade (para manter os custos) de forma a conseguir manter o prazo de entrega, com o mesmo número de pessoas envolvidas.

2.1.1. Atividades, ações e tarefas no desenvolvimento de Software

Existe um conjunto de atividades, ações e tarefas associadas ao desenvolvimento de *Software* que são comuns a todos os modelos existentes. (Pressman 2010).

- Uma atividade procura alcançar um objetivo amplo (por exemplo, a comunicação com as partes interessadas), é aplicada independentemente do domínio da aplicação, tamanho do projeto, esforço e complexidade exigidas ou o grau de rigor exigido do Software devido a situação onde irá ser aplicado ou operar (Grau de criticidade).
- Uma ação (por exemplo, o design do sistema/software) engloba um conjunto de tarefas que produzem uma parte principal dos trabalhos (por exemplo, que produzem a modelagem do projeto).
- Uma tarefa procura atingir um objetivo pequeno, mas bem definido (por exemplo, conduzir a um teste unitário) que produz um resultado tangível.

Cada modelo implementa no entanto de uma forma diferente estas atividades, ações e tarefas, dando por exemplo, mais ou menos enfase a algumas delas, ou executando por exemplo algumas destas em simultâneo. Pressman define 5 atividades genéricas, no processo de desenvolvimento de *Software*. (Pressman 2010)

- **Comunicação.** Antes de qualquer trabalho mais técnico, é necessário uma fase de comunicação e colaboração entre cliente e restantes Stakeholders no sentido de ser definidos os objetivos para o projeto (Âmbito), de forma a iniciar o processo de recolha de requisitos, de forma a definir as características e funcionalidades desejadas (Nos modelos mais recentes esta colaboração é feita ao longo de todo o projeto).

- **Planificação (**Analise**).** Qualquer viagem complicada fica muito mais simplificada se existir um mapa. A planificação elabora um mapa através do estudo do problema, análise dos requisitos, e dos potenciais riscos. De forma a conseguir definir o trajeto, as tarefas e os recursos necessários para chegar ao destino com sucesso.

- **Modelagem** (*Design)*. A criação de modelos ajuda na definição de uma visão mais ampla e abrangente do problema. Esta fase é a da concetualização do *Software,* onde se procura perceber como todas as peças encaixam e as suas características específicas. Pode ser necessário proceder a um refinamento de partes do modelo, no sentido de obter um maior detalhe que ajude a perceber qual a melhor forma de o resolver questões (duvidas) mais específicas.

- **Construção** (Implementação). É nesta fase que é gerado o código, de uma forma manual ou mais automatizada. Após a criação do código procede-se à execução de testes, de forma a descobrir a possível existência de erros, e caso os haja procede-se à correção dos mesmos. (Os testes por vezes constitui uma atividade separada).

- **Entrega** (*Deployment*). Nesta fase é feita a entrega do *Software* ao cliente, de uma forma completo ou parcial (entrega incrementalmente), de forma a estes poderem avaliar e dar *Feedback* do que foi entregue. Associada a esta atividade, estão as tarefas suporte, manutenção e treino dos utilizadores.

Para além destas 5 atividades principais, podemos considerar um conjunto de atividades auxiliares mas transversais as 5 atividades definidas acima, como o controlo e rastreio de projeto, a gestão de risco, revisões técnicas, garantia de qualidade e gestão da configuração.

Existe uma atividade que é transversal a todas as outras.

A Documentação. Durante todo o processo de desenvolvimento, existe a necessidade de se criar documentação para vários fins. Podemos identificar 4 tipos distintos de documentação (Sommerville 2010):

- Que é produzida para dar suporte durante o período de desenvolvimento.
- Que é destinada a quem utiliza o *Software.*
- Que é destinada a quem dá suporte ao *Software /* manutenção.
- Que é destinada em quem tem de adaptar, alterar ou estender as capacidades do *Software*.
- Que é destinada para fins de Marketing e estudo de mercado.

2.1.2. Desenvolvimento de Software - uma tarefa difícil

Existe uma perceção generalizada de que o desenvolvimento de *Software* é uma tarefa difícil. Quais serão os fatores principais que influenciam tal perceção?

Brooks identifica 4 fatores principais, a complexidade, a conformidade, a mutabilidade e a invisibilidade (Brooks 1995):

A complexidade.

Está na essência do desenvolvimento de *Software*. Esta torna difícil concetualizar na íntegra um projeto. Dificulta a comunicação entre os intervenientes, o que pode levar ao aparecimento de falhas nos produtos, custos excessivos, e atrasos nos prazos de entrega. Torna difícil a enumeração e perceção de todos os estados possíveis de uma aplicação, tornando-se por isso pouco fiável. Dificulta alterações ou extensões da mesma sem o aparecimento de efeitos secundários indesejáveis. Faz com que existam curvas de aprendizagem e entendimento lentas, o que pode ser potencialmente desastroso quando existe a saída de algum interveniente a meio do desenvolvimento.

A conformidade

Quem desenvolve tem normalmente de adaptar o que vai desenvolver ao que já existe. Não existe nenhum princípio regulador da forma como isto pode ser feito, cada caso é único. Cada instituição e sistema terão necessidades de interfaces diferentes, não existindo por isso forma de simplificar esta tarefa.

A mutabilidade

Num edifício já construído, qualquer posterior modificação estrutural, pode não ser implementada porque existe uma clara perceção de todos os envolvidos, de que os custos associados tornem esta modificação inviável. Pelo contrário no caso do *Software*, existe uma pressão constante no sentido deste se adaptar a novos ambientes, para além do domínio originalmente definido. O utilizador poderá por exemplo, identificar mudanças nas funções existentes no sentido de melhorar as mesmas ou identificar a necessidade de funções inexistentes, poderá existir a necessidade do *Software* ter de se adaptar a novo *Hardware.*

A invisibilidade

Procuramos representar visualmente uma aplicação através de vários diagramas, de forma a conseguir concetualizar algo que ainda não existe na realidade. Mas estas representações não são, pelo menos para o cliente, fáceis de interpretar, no sentido de através das mesmas, conseguir "visualizar" o resultado final. Este facto prejudica seriamente a comunicação entre os intervenientes.

2.2. Tipos de modelos de desenvolvimento

Podemos distinguir os vários modelos pela forma diferente em como os processos são executados em cada um destes (Pressman 2010):

Linear - Diz-se que um modelo é linear quando as atividades são executadas sequencialmente uma apos a outra. (figura 2)

Figura 2 - Fluxo linear de atividades (Processos)

Iterativo - Diz-se que um modelo é iterativo quando uma ou mais atividades são repetidas antes de ser retomada a próxima atividade (em sequencia). (figura 3)

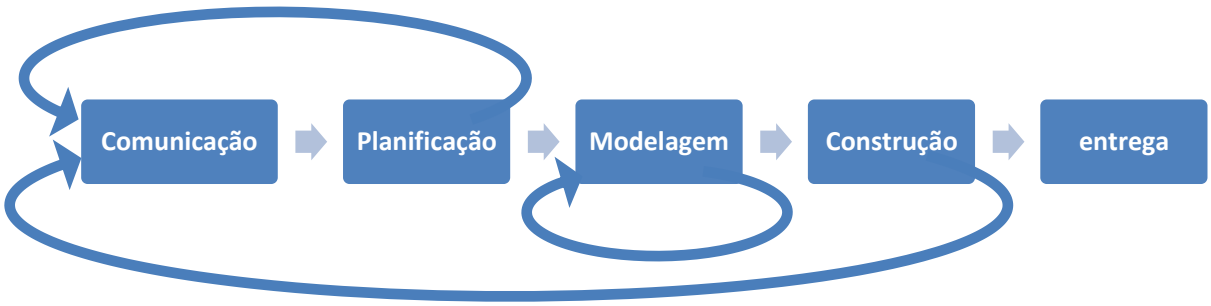


Figura 3 - Fluxo iterativo de atividades (Processos)

Evolucionário - Diz-se que um modelo é evolucionário quando existe um "fluxo circular" das atividades. Em que a cada ciclo de atividades (fluxo) se obtém uma versão mais completa do *Software*. (figura 4)

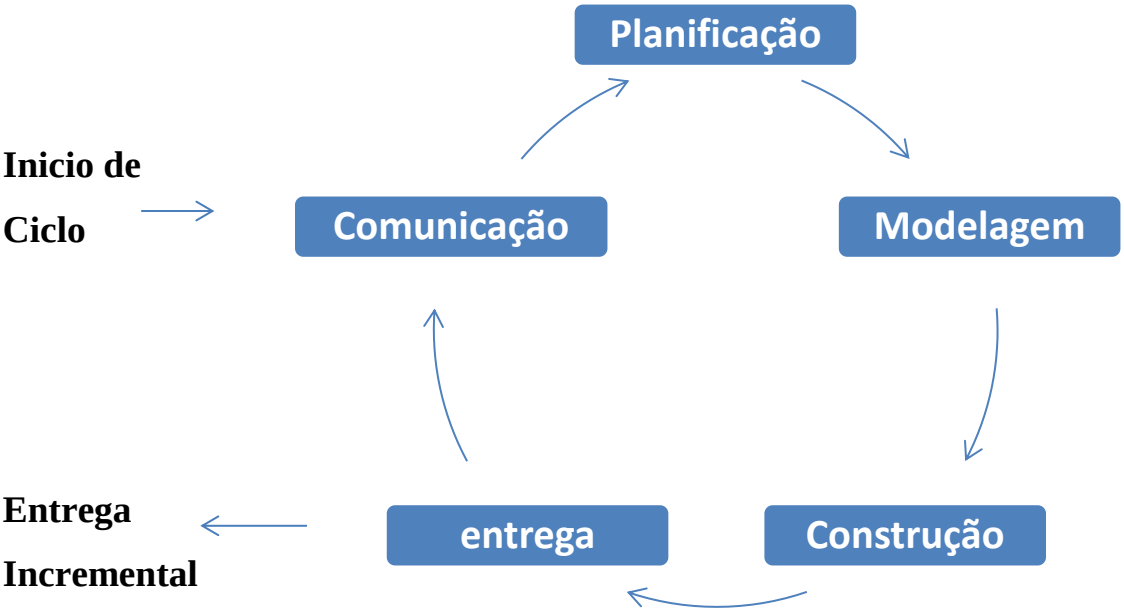


Figura 4 - Fluxo evolucionário de atividades (Processos)

Paralelo - Diz-se que um modelo é Paralelo quando existe mais do que uma atividade a ser executada ao mesmo. (figura 5)

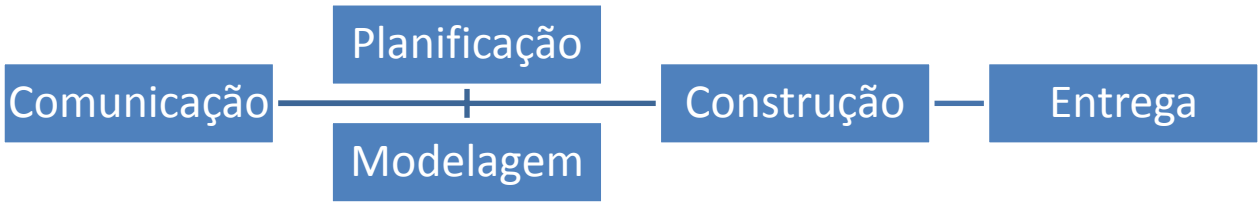


Figura 5 - Fluxo paralelo de atividades (Processos)

2.2.1. Desenvolvimento tradicional (Modelo Cascata/Waterfall)

O método tradicional de desenvolvimento de *Software* é do tipo linear quando aplicado no seu estado "puro" (mas pouco natural), apenas se segue para a próxima fase, depois de concluída a fase anterior. Revisões podem ocorrer antes da passagem à fase seguinte, o que possibilita a incorporação de mudanças, que pode envolver um processo de controlo de mudança formal (documentação). Revisões podem também ser utilizadas para garantir que determinada fase está de facto concluída. Normalmente existe a ideia de que neste modelo se desencoraja o regresso a qualquer fase prévia uma vez concluída, o que não corresponde à realidade. (Royce 1970)

Este modelo (Sommerville 1996; Royce 1970)é composto pelas seguintes fases:

- Elicitação dos requisitos e documentação "muito completa" dos mesmos.
- Análise de requisitos.
- *Design Software.*
- Criação de código (implementação e integração).
- Testes (validação).
- Operações (implantação ou instalação, manutenção).

Neste tipo de abordagem, a documentação "completa" prevista, trás consigo 2 problemas principais (Eckstein & Baumeister 2004):

- O primeiro problema é o facto de a documentação só ter algum valor enquanto estiver atualizada, o que pode ser um desafio. Deve estar em conformidade com todas as alterações feitas nas várias fases.

- O segundo problema relaciona-se com o facto de esta abordagem exigir a produção de um grande volume de documentação muito detalhada, elaborada pelos vários intervenientes, tendo por isso de seguir uma certa padronização de forma a poder ser interpretada corretamente por todos os interessados.

- Os autores desta documentação poderão ter tendência de omitir alguma informação mais sensível ou duvidosa, por uma questão de precaução ou reserva, pelo facto destes documentos ficarem disponíveis a todos os interessados.

A aparente "rigidez" existente, pelo menos no modelo *Waterfall* "puro", para além da necessidade de criar muita documentação, por muitos considerada excessiva, faz com que este modelo tenha vindo a ser muito criticado por apoiantes de outros modelos, (ditos) mais "flexíveis".

Este modelo poderá funcionar "bem" em projetos onde não haja "dúvidas" em relação aos requisitos, onde os mesmos não tiverem, por isso, de sofrer alterações. (Boehm 1987)

Exemplos deste tipo de projetos são os casos de (entre outros):

- Sistemas de controlo Aéreo,
- *Software* de naves espaciais

Esta abordagem é normalmente também designada de *Plan-driven development* ou seja, um desenvolvimento que é impulsionado por um plano. Mas será que o plano funciona? A questão passa um pouco por ai… talvez dependa do plano.

Goug Hall afirma que "A percetiva clássica é de se tentar definir corretamente 95% de um plano de negócios ou de produto, antes de tomar qualquer outra ação… este processo …em teoria parece ser excelente, porém… na realidade …raramente resulta. Porquê? Porque assim que o produto chega ao mercado, reconhecemos imediatamente as suas falhas fatais. Nesse momento, constatamos muitas vezes que já não vamos a tempo de corrigir o produto a embalagem ou a forma como o comercializamos". (Hall 1987)

Quase 40 anos antes, Royce afirma exatamente o mesmo, em relação a este modelo de desenvolvimento tradicional de *Software*, a quem muitos atribuem erradamente a sua autoria (mesmo a nível académico). É de facto Boehm, que muitos anos mais tarde, Atribui o nome de *Waterfall* a este tipo de modelo. (Boehm 1988)

Na realidade o conceito por detrás do *Waterfall* parece surguir primeiro em 1956, pelas mãos de Benington numa apresentação feita por este, num simpósio intitulado "Métodos avançados de programação para computadores digitais", patrocinado pela Marinha Americana. (Benington 1987) Conforme podemos constatar no fluxograma apresentado abaixo.

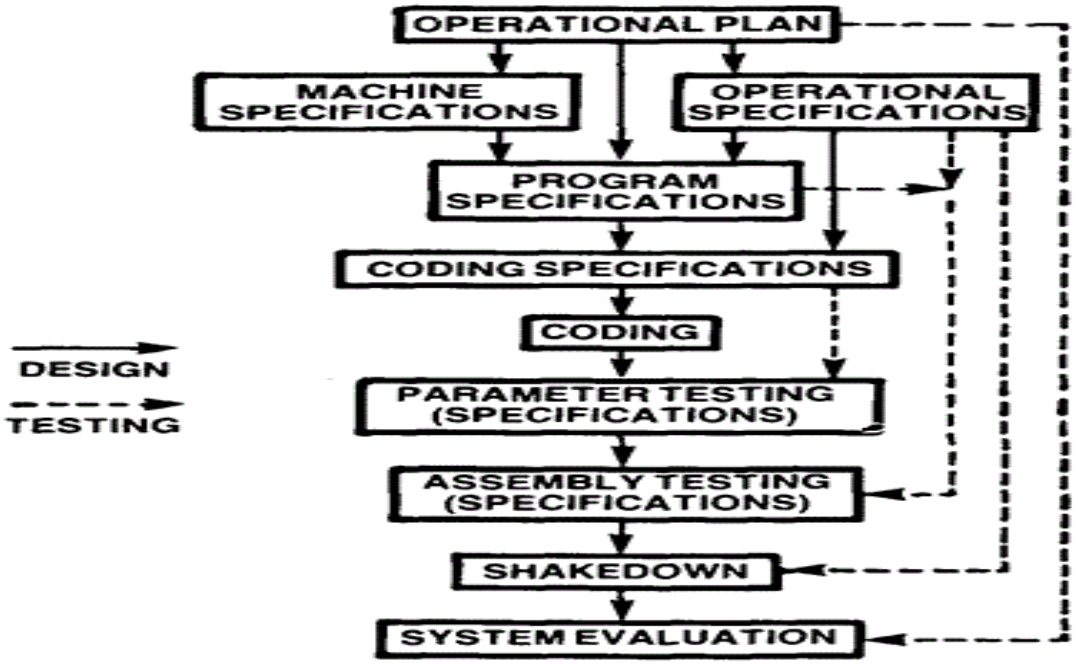


Figura 6 - Representação de Benington do processo de desenvolvimento de software tradicional (Benington 1987)

Benington, no seu artigo original, não faz qualquer referência à prototipagem. No entanto este afirma no artigo adaptado mais tarde, que já na década de 50 do século passado, no âmbito do trabalho por este realizado no sistema de proteção anti areio da Força Aérea dos Estados Unidos da América (SAGE - *Semi-Automatic Ground Environment*) se recorria extensivamente ao uso de prototipagem numa fase inicial do desenvolvimento, antes de se proceder ao desenvolvimento de *Software* em grande escala. (Boehm 1987)

Royce apresenta um fluxograma, muito similar ao anterior, de Benington. Mas o facto é que este afirma na linha logo a seguir ao fluxograma, que "acredito no conceito (em teoria), mas a implementação descrita acima é ariscada e é um convite ao fracasso". (Royce, 1970).

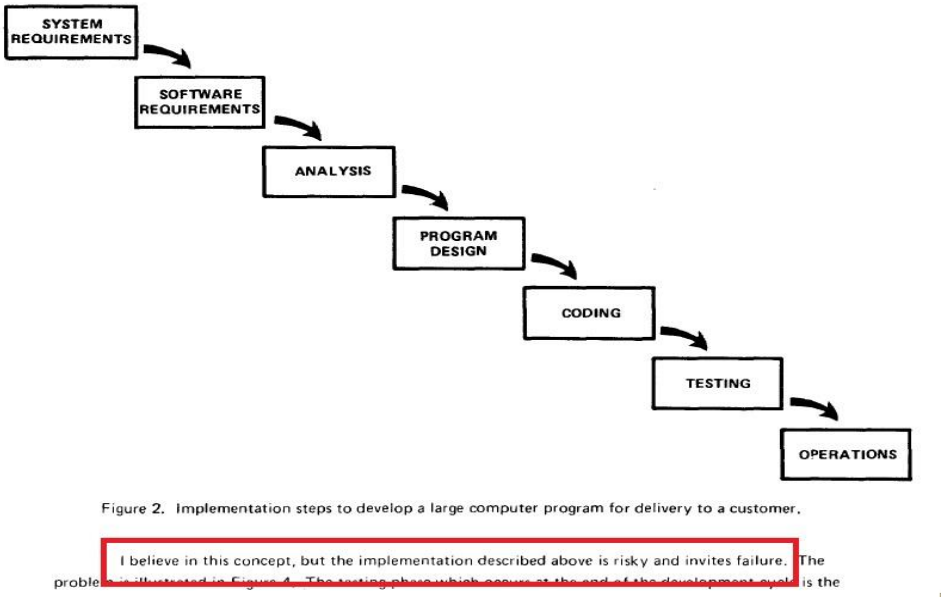


Figura 7 - Representação de Royce do processo de desenvolvimento de software tradicional (Royce 1970)

Royce não só não criou o modelo *Waterfall*, como também não defendeu, no seu artigo, a sua aplicação pelo menos no seu estado "puro". No entanto, durante décadas este modelo foi um dos mais usados para o desenvolvimento de *Software*. Embora o modelo em cascata original proposto por Royce preveja a execução "*Loops* de feedback", a grande maioria das organizações que aplicaram este modelo trataram no como se este fosse estritamente linear. (Royce 1970; Pressman 2010)

Infelizmente, quem foi adotando este modelo ao longo de décadas, muito em função de pressupostos criados a partir do fluxograma fatídico apresentado na segunda página do artigo de Royce, certamente nunca procurou ler ou teve acesso ao artigo do pseudoautor do modelo *Waterfall*. Bastava ler a linha a seguir para perceber que Royce defendia que o modelo tal como apresentado no fluxograma, não funcionaria da melhor forma salvo raras exceções, por exemplo quando os requisitos estão completamente definidos e são imutáveis. O que sabemos hoje ser bastante improvável.

2.2.2. Os conceitos de desenvolvimento iterativo e incremental

Estes Conceitos surgem segundo Larman e Basili, em 1939 com o livro de Walter Shwhart, perito em qualidade da Bell Labs. Este propôs a execução de uma série de pequenos ciclos de planificação, execução, estudo (verificação) e ação (agir), do inglês *Plan-Do-Study-Act*, no sentido de se obter melhorarias de qualidade do produto final. A partir do início da década de 40, W. Edwards Deming é um dos maiores defensores e impulsionadores do uso deste conceito, também conhecido como de melhoria continua, que muitos identificam com o conceito *Kaizen*, que não é mais do que a implementação realizado por Deming do *Plan-Do-Study-Act* adaptado à realidade e cultura do Japão. Segundo Larman e Basili, um dos primeiros projetos de *Software* a ser desenvolvido, através do conceito de desenvolvimento iterativo e incremental foi o desenvolvimento de *Software* para o "Projeto Mercury", este foi o primeiro projeto tripulado de exploração espacial da NASA, começado no início da década de 60. (Basili & Larman 2003)

2.2.2.1. Desenvolvimento iterativo

O desenvolvimento iterativo caracteriza-se pela repetição de algumas atividades durante o processo de desenvolvimento. Royce não inventa o desenvolvimento iterativo, no entanto é um dos primeiros a afirmar a necessidade de o utilizar, quando na 3ª figura de Royce, este demonstra a necessidade de existência de um caracter iterativo na relação entre as fases sucessivas de desenvolvimento, a fim de se conseguir corrigir dificuldades imprevistas detetadas *a posteriori*. (Royce, 1970)

Em relação à 4ª figura de Royce, ele afirma que mudanças de *Design* necessárias para corrigir um problema detetado posteriormente, serão provavelmente tão disruptivas, que certamente ficarão a violar os requisitos iniciais em que os erros ou omissões assentavam, nunca se ficando por isso as iterações apenas restringidas as fases sucessivas. (Royce, 1970)

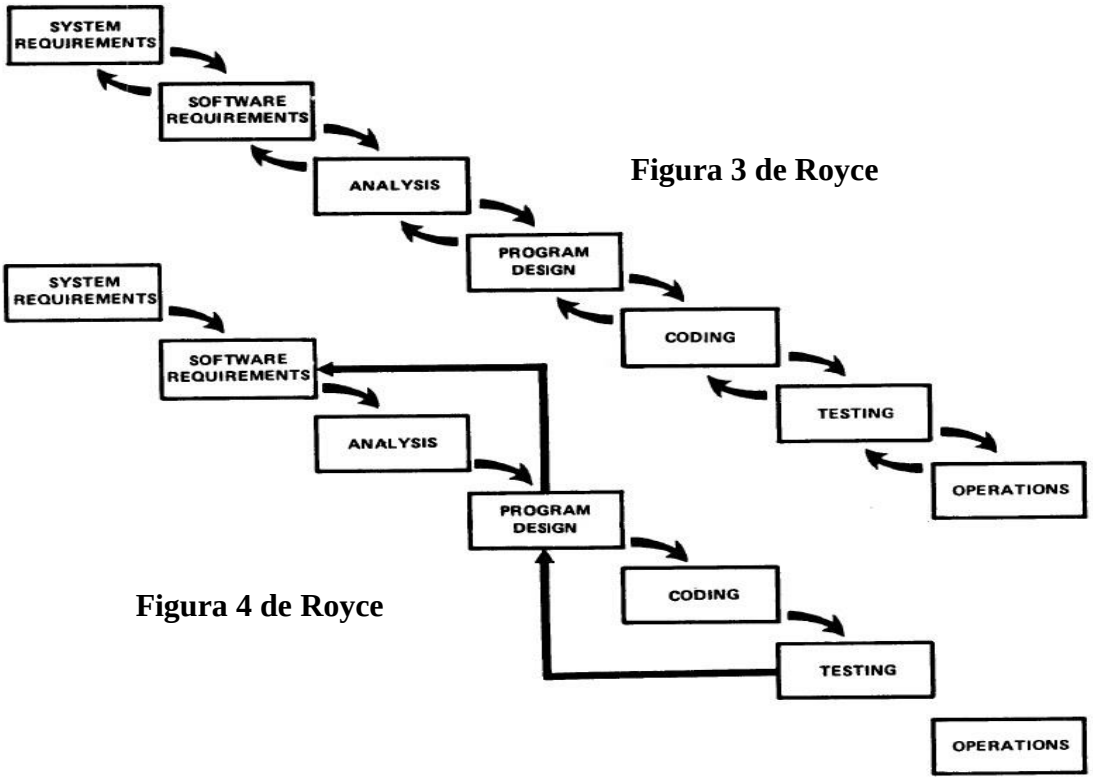


Figura 8 - Representação de propostas de alterações ao processo de desenvolvimento de Software tradicional (Royce 1970)

Neste caso Royce afirma que poderá existir a necessidade de se fazer mudanças ao nível dos requisitos, ou mudanças substanciais ao nível de *Design*. Em resultado disto existe a possibilidade de termos de voltar à origem do problema, o que faz com que seja espectável um aumento de até 100% em termos de custos e tempo gasto com o projeto.

A iteração sucessiva do *Waterfall* por si só, não trás grandes vantagens a não ser conseguirmos eventualmente obter um produto que esteja mais de acordo com o que era inicialmente espectável, ao fim de umas quantas tentativas (iterações).

2.2.2.2. Desenvolvimento incremental

Desenvolver *Software* através da entrega incremental, requer antes de mais, que seja feita uma previsão inicial (esqueleto) do que deve de ser a implementação final, fornecendo uma implementação simples das funções de operação essenciais. Posteriormente, através de várias iterações, devem ser proporcionadas aos utilizadores, versões mais capazes e melhoradas do sistema, em intervalos regulares, até se alcançar uma implementação (final), que corresponda a todos os requisitos do projeto. (Basili & Turner 1975) Existe uma redução do risco inerente a este tipo de desenvolvimento, devido ao facto de ser feita uma segmentação do problema maior, em vários ciclos ou partes mais pequenas. Esta segmentação facilita a introdução de possíveis correções nos ciclos seguintes. (Medicare et al. 2012)

2.2.3. Conceito de Prototipagem

No seguimento dos conceitos de desenvolvimento iterativo e incremental, surge um outro conceito que está ligado a estes, que é o conceito de prototipagem.

Um protótipo tipicamente simula alguns dos aspetos do Software pretendido. No sentido em que estes podem ajudar a reduzir o fosso que possa existir entre as visões do cliente e de quem desenvolve, as quais que tendem a ser diferentes. Através do uso dos vários tipos de protótipos conseguimos colmatar lacunas de entendimento que possam existir em relação ao que deve de ser o produto final. O cliente ao contactar com uma parte do novo sistema ou um esboço do mesmo, pode ajudar a clarificar os requisitos. Sendo até certamente mais divertido faze-lo desta forma, do que tentar clarificar seja o que for através da leitura enfadonha de documentação bastante técnica.(Sommerville 2010; Pressman 2010)

Através da observação da interação de utilizadores com os protótipos conseguimos perceber, se o produto que está a ser desenvolvido com base nestes (protótipos) irá realmente satisfazer as suas necessidades. Conseguimos um feedback mais atempado, de forma a chegarmos mais rapidamente a um entendimento comum do que devem ser os requisitos do *Software* (validação dos mesmos), o que ajuda a reduzir o risco potencial de insatisfação do cliente em relação ao produto final.

O guru do marketing Doug Hall criou o termo, que hoje está muito em voga, "Fail Fast, Fail Cheap" (Hall 1987), ou seja falhar rápido é falhar barato. A prototipagem permite a quem desenvolve, "falhar numa pequena escala" para conseguir "ter sucesso em grande escala. (Bernstein 1996). O ex-patrão da Procter & Gamble (P&G), Alan Lafley, afirma que "aprendemos muito mais com o fracasso do que com o sucesso". (Lafley 2011)

De facto conforme diz o ditado popular …"o sucesso provem do bom discernimento, o bom discernimento surge da experiencia, a experiencia provem do mau discernimento"

Sabemos hoje que erros detetados precocemente através do recurso à prototipagem ajudam a produzir um produto final melhor, para além de ajudar a reduzir o risco de falha do projeto, seja quanto:

- Aos custos
- Tempo gasto com o mesmo
- Produto final insatisfatório.

2.2.4. Modelo em V

Este modelo é muito semelhante ao modelo tradicional, o que o torna diferente, é a existência de uma maior ênfase na execução de testes. Existindo normalmente 2 equipas distintas, cada uma dedicada a um dos ciclos, Existe portanto 2 ciclos distintos são executados em paralelo (Pressman 2010):

- O ciclo de desenvolvimento (verificação)
    - Análise de requisitos.
    - Especificação funcional.
    - *Design* de alto-nível.
    - *Design* detalhado / especificações de programa.

- O ciclo de testes (validação)
    - Testes de aceitação do utilizador.
    - Testes de sistema.
    - Testes de integração.
    - Teste de unidade.

Os benefícios deste modelo são fáceis de perceber. Existe uma relação entre cada fase de ambos os ciclos, no sentido de apenas se proceder à passagem para uma fase de desenvolvimento seguinte após terem sido executados as validações necessárias (testes) para a mesma fase.

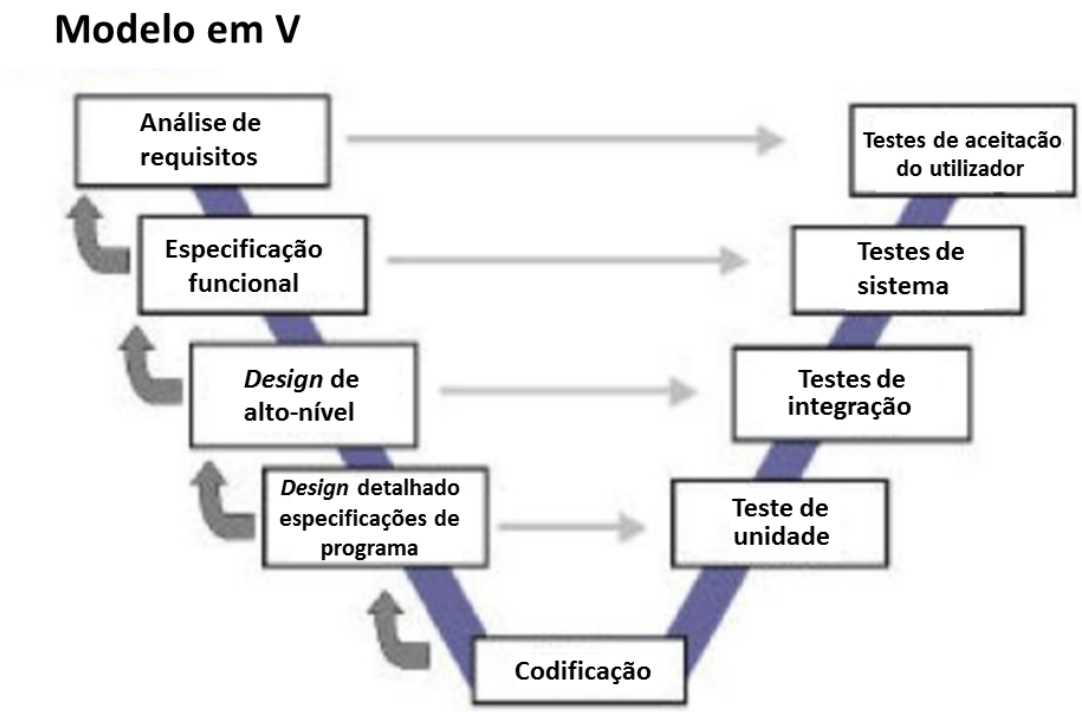
Este modelo ajuda a identificar precocemente as discrepâncias que possam existir. Reduz o custo de correção de erros devido à deteção precoce dos mesmos.

Neste modelo, existe uma convergência entre os processos dos ciclos de desenvolvimento e testes, desde o início até ao fim de ambos. Ou seja, a equipa de desenvolvimento tenta implementar a solução, enquanto a equipa de testes desenvolve em paralelo processos para eliminar ou minimizar o risco.

As equipas de testes e desenvolvimento trabalham para reduzir a ocorrência de erros e aumentar a aceitabilidade do produto final.

O modelo em V consiste em 9 fases. As 4 fases de verificação estão do lado esquerda do V, a fase de codificação está na parte inferior do V e as 4 fases de validação que estão do lado direito do V.

Figura 9 - Representação do processo de desenvolvimento do modelo em V[7]



[7] Adaptado a partir de fonte obtida em:
Http://sqa.fyicenter.com/FAQ/Software-Development-Models/Software_Development_Models_V_Model.html

2.2.5. Modelo em Espiral

Segundo Boehm, o modelo em espiral tem uma abordagem diferente dos modelos anteriores, à qual ele designa *Risk-driven development* ou seja, um desenvolvimento que é impulsionado, pela procura em mitigar os riscos. Incorpora muitos dos pontos fortes dos outros modelos e procura resolver muitas das suas dificuldades. (Boehm 1988)

O modelo em Espiral é incremental e iterativo, mas ao invés de se limitar a repetir sequencias repetidas do waterfall procurar ativamente identificar, analisar e reduzir riscos no processo de desenvolvimento, através do uso de protótipos numa fase inicial e do processo iterativo, no sentido de conseguir validar o mais precocemente possíveis requisitos não identificados inicialmente ou em que existe alguma incerteza quanto aos mesmos.

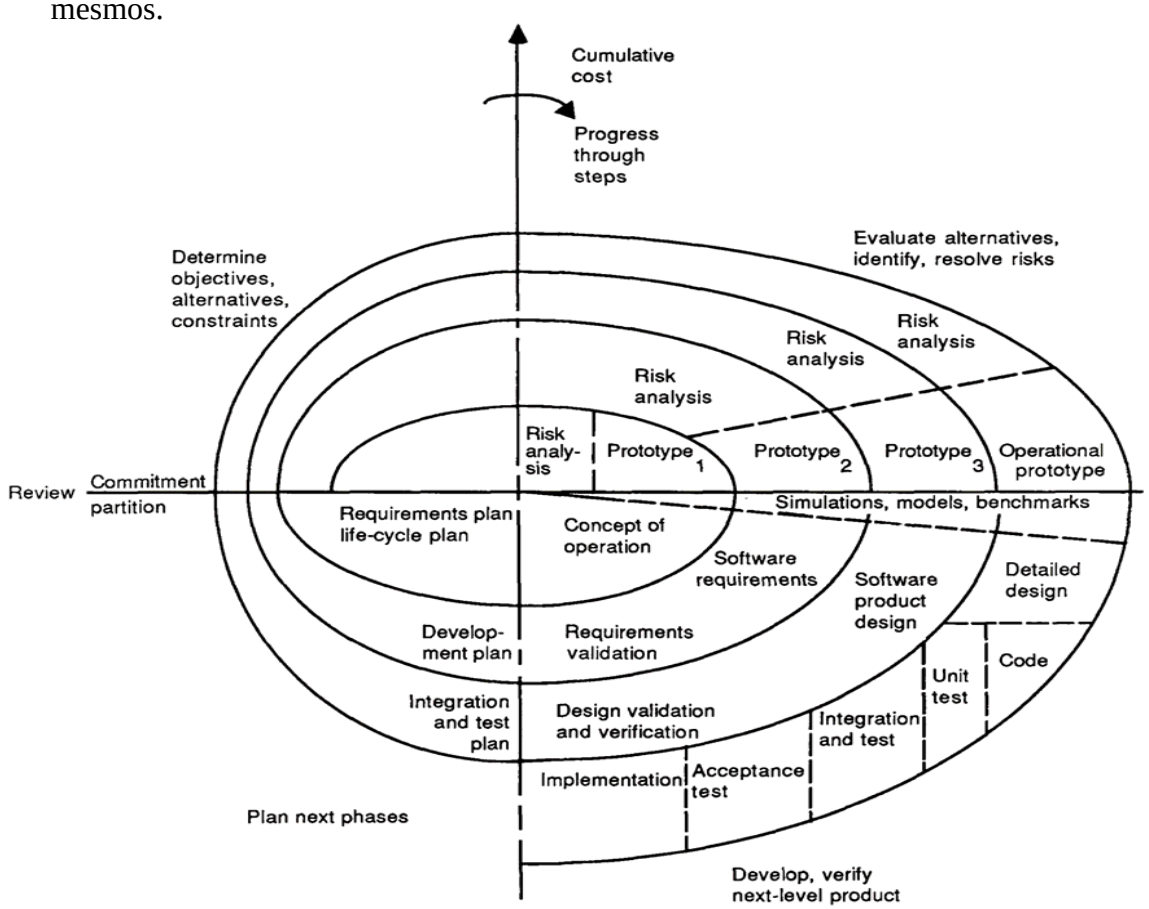


Figura 10 - Representação de Benington do processo de desenvolvimento do modelo em espiral (Boehm 1988)

2.2.6. Rapid Application Development (RAD)

O conceito por detrás do desenvolvimento rápido de aplicações (RAD) foi formalizado por James Martin, no seu livro de 1991 com o mesmo nome "*Rapid Application Development*". (Martin 1991) O RAD é um conjunto integrado de técnicas, orientações e ferramentas que facilitam a criação e implementação mais rápida de *Software* que consiga satisfazer necessidades do negócio do cliente, num período curto de tempo. Caracteriza-se por isso, por existir um grande envolvimento dos utilizadores futuros do *Software* em todas as fases do desenvolvimento, que é feito de forma incremental e iterativo.

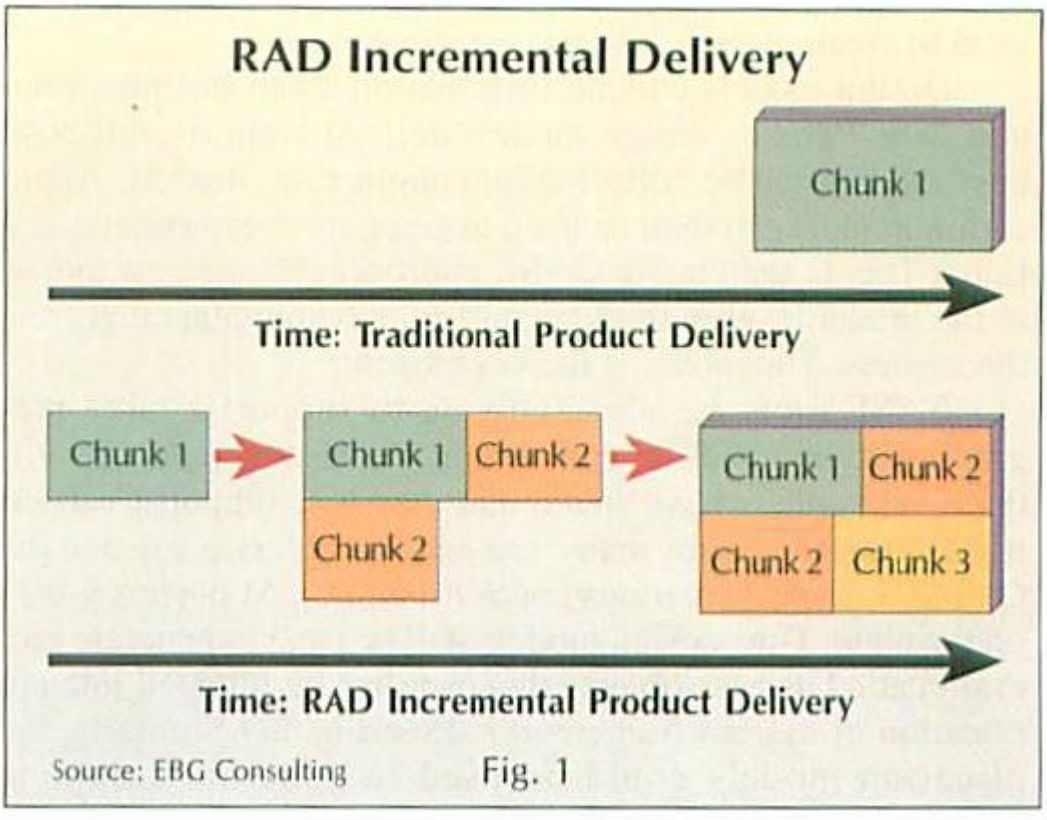


Figura 11 - Comparação entre RAD e Waterfall
Entrega incremental (Gottesdiener 1995)

O *Software* não "aparece" apenas no final do ciclo de desenvolvimento. Em vez disso, vai evoluindo durante todo processo de desenvolvimento RAD com base no feedback contínuo do cliente. O *Software* não é entregue de uma só vez. Este é entregue em pedaços por ordem de importância para negócio.

Surge o conceito de priorização no desenvolvimento das várias "funções" a integrar na aplicação, em função do valor que cada uma tem para o negócio. A implementação do *Software* é feita de forma faseada, pouco a pouco. Este conceito, tem atualmente um valor de destaque em todos os modelos mais recentes. (Gottesdiener 1995)

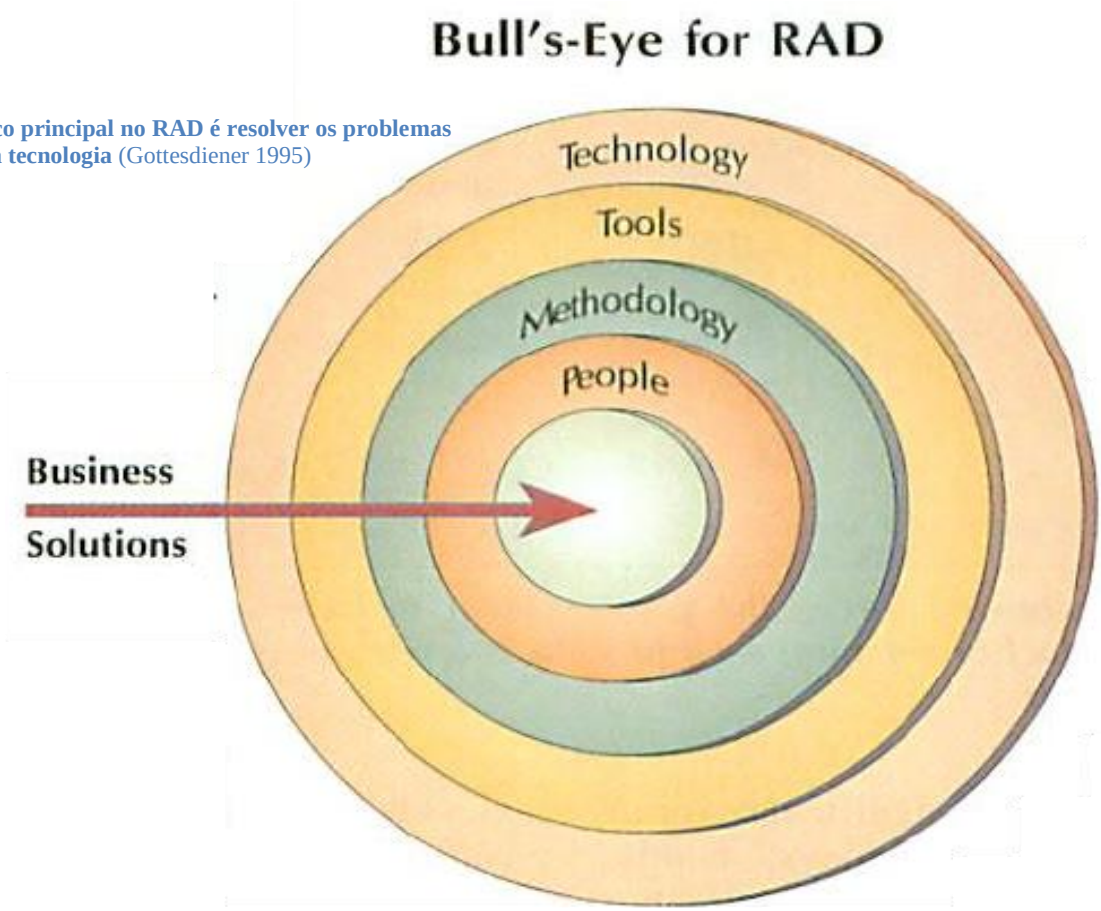


Figura 12 - O foco principal no RAD é resolver os problemas de negócio, não a tecnologia (Gottesdiener 1995)

2.2.7. Desenvolvimento "Agile" de Software

Em 2001 surge o "*Manifesto Agile*", os seus autores pretenderam através deste, estimular mudanças no panorama do desenvolvimento de *Software* de então. Este modelo, teve um início humilde, mas ao longo dos últimos 13 anos, tem vindo cada vez mais a se adotado pela industria, podendo ser até considerado como o modelo padrão atual. (Fowler & Highsmith 2001)

Este manifesto afirma o seguinte[8]:

Estamos a descobrir melhores maneiras de desenvolver *Software*, fazendo-o nós mesmos e ajudando os outros a fazê-lo. Através deste trabalho, passamos a valorizar:

- **Indivíduos e interações,** mais do que processos e ferramentas
- ***Software* que funcione,** mais do que documentação abrangente
- **Colaboração com o cliente,** mais do que negociação de contratos
- **Responder a mudanças,** mais do que seguir um plano

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

Podemos considerar, de certa forma o *"Agile"* como uma filosofia, por esta pretender defender valores e princípios, para além de procurar também definir diretrizes gerais para o desenvolvimento de *Software.* Esta filosofia encoraja a satisfação do cliente, através participação e comunicação ativa e continua entre todos os envolvidos. Procura fazer uma entrega incremental de *Software* Funcional, recorre para isso, a equipas pequenas de desenvolvimento mas altamente motivados. (Pressman 2010).

[8] Agile Manifesto - http://agilemanifesto.org/

Os Autores do "*Manifesto Agile"* definem os 12 seguintes princípios:

- A nossa maior prioridade é a satisfação do cliente através da entrega rápida e contínua de *Software* valioso.
- Acolhemos com agrado alterações de requisitos, mesmo perto do final de desenvolvimento. As mudanças feitas através dos processos ágeis ajudam o cliente obter maior vantagem competitiva.
- Entrega *Software* que funcione, com frequência, que pode ir de um par de semanas a alguns meses, com preferência para escalas de tempo mais curtas.
- Clientes e equipa de desenvolvimento devem trabalhar juntos diariamente ao longo do projeto.
- Deve de se construir projetos em torno de indivíduos motivados. Dar-lhes o ambiente e o apoio que necessitam e confiar neles para fazer o trabalho.
- O método mais eficiente e eficaz de transmitir informações para dentro de uma equipa de desenvolvimento é a conversa cara a cara.
- A principal medida de progresso é a funcionalidade do *Software*.
- Processos ágeis promovem o desenvolvimento sustentável. Os patrocinadores, equipa de desenvolvimento e utilizadores devem de ser capazes de manter um ritmo constante indefinidamente.
- Atenção contínua à excelência técnica e bom *Design* aumenta a agilidade.
- Simplicidade é a arte de maximizar a quantidade de trabalho que não é necessário… isto é essencial.
- As melhores arquiteturas, requisitos e projetos emergem de equipas que são auto-organizadas.
- Em intervalos regulares, a equipa deve refletir sobre como tornar-se mais eficaz. De seguida, deve de proceder aos ajustamentos de comportamento necessários de forma a corresponder ao que foi anteriormente refletido.

Em contra ciclo com os modelos associados ao "Plan-driven", surgem modelos mais agueis, que estão mais focados nos processos de desenho iterativo e incremental. Nestes o código surge mais cedo. Através destes modelos, procura-se obter um feedback mais precoce dos utilizadores.

No entanto Estes Modelos não são propriamente um corte com tudo o que foi feito no passado, surgem mais como uma tentativa de correção dos erros cometidos no passado, próprios e dos outros. Mas expor os erros que possamos ter tido no passado, em função de erros contidos em modelos de desenvolvimento criados por outros, é muito mais fácil, do que ter um espirito critico em relação a si mesmo. Quando se foi adotando os modelos clássicos, assentes em fundamentos e interpretações que não correspondiam na realidade ao que os (pseudo) autores realmente defendiam, de quem é a culpa. Talvez não seja de ninguém e de todos.

Isto porque muitos dos erros cometidos ao longo dos anos foram cometidos por falhas de comunicação e interpretação do que os autores de esses modelos preconizavam verdadeiramente. E se há uma certa culpa por parte de quem interpreta também existe alguma culpa por parte de quem comunica.

Felizmente como o ditado popular diz:

"Nada se perde, tudo se transforma".

Através da experiencia do uso destes modelos clássicos, conseguiu-se expor, as falhas que ainda possam persistir, e potenciar mudanças que possibilitem melhores resultados nos modelos futuros.

Se calhar tudo poderia ter sido evitado se Royce no seu Artigo mais (ou talvez menos) conhecido tivesse procurado ser mais claro do que eu acho que foi, apresentado o fluxograma problemático da seguinte forma:

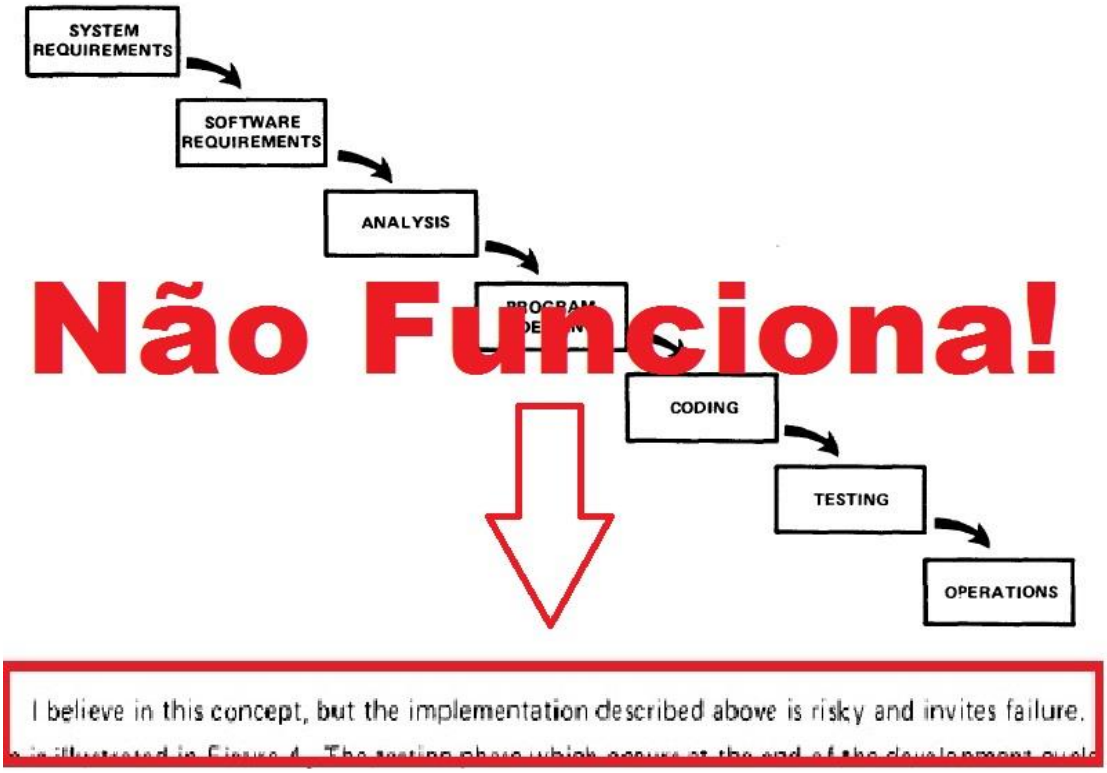


Figura 13 - Adaptação feita ao fluxograma original do autor (Royce 1970)

Quem teve a capacidade de não ler para além do fluxograma de Royce e tomou logo partido pelo dito modelo "Puro", não viu Royce defender o uso da iteração, no texto e fluxogramas seguintes. Porque para além disso, este propõe no mesmo artigo mais um conjunto de conceitos essenciais, dos quais podemos destacar 2 deles por terem já há quase 45 anos um caracter bastante "ágil":

O primeiro conceito é o "DO IT TWICE", ou seja de fazer o processo todo 2 vezes (pelo menos), como ele diz, "através da construção de um "modelo piloto" de deitar fora"… feito em paralelo ao processo todo …"de forma a explorar (precocemente) elementos novos e fatores desconhecidos". (Royce, 1970) Estava-se a referir, de certa forma já naquela época, ao uso de protótipos.

O segundo conceito é o do envolvimento/colaboração do cliente em todas as fases do processo. Royce afirma mais concretamente que "por alguma razão, mesmo depois de os envolvidos terem chegado a um acordo, na fase de *Design*, do que o *Software* tem de fazer, através dos requisitos, estes estão sujeitos a interpretações muito alargadas (diferentes). É por isso, importante envolver o cliente de uma forma formal, para que este fique comprometido nas fases anteriores à entrega final. Dar "rédeas soltas" ao contratante entre a fase de definição de requisitos e a fase de operações é estarmos a pedi-las".(Royce 1970)

Segundo Wiegers, o fator mais critico para a qualidade do *Software* (sucesso), é o envolvimento/colaboração do cliente. Qualquer empresa de sucesso (deve de) preocupa-se não só, que um colaborador seu consiga fazer as coisas corretamente, mas também que este consiga fazer o que está certo.(Wiegers 1996)

No caso do desenvolvimento de *Software*, espera-se que a equipa de desenvolvimento saiba fazer as "coisas" bem (tenham conhecimento sólidos/capacidades técnicas suficientes).

Mas isso não é suficiente. Fazer o que está certo, requer uma compreensão inequívoca do que o cliente espera, o que ele pretende ou precisa. O que por vezes é bastante problemático conseguir perceber, porque nem o cliente sabe o que quer. Nesse sentido devemos ambicionar o máximo de participação dos clientes/utilizadores em todas as atividades do desenvolvimento.(Wiegers 1996)

É curioso, que 30 anos antes do aparecimento do manifesto "Agile", Royce já defendia o que sabemos hoje ser uma das bandeiras desse manifesto, o envolvimento / colaboração do cliente ao longo de todo o processo de desenvolvimento. No entanto a única coisa que pareceu sobressair do seu artigo foi o fatídico fluxograma.

2.3. Abordagens distintas

Podemos distinguir essencialmente 2 abordagens diferentes no desenvolvimento de *Software*. Apesar de se poder optar por uma ou por outra, sabe-se hoje que não existe de facto uma forma única ou correta de desenvolver *Software*. A solução passa mais pelo uso de uma das várias receitas disponíveis, as quais são compostas por doses diferente de ambas as abordagens. (Bernstein 1996)

2.3.1. Uma abordagem Top-Down

Podemos definir esta abordagem como sendo de ***análise***, ou seja "é o processo de decomposição de um tópico complexo, nos seus diversos elementos constituintes… a fim de se obter uma melhor compreensão sua". (Wikipedia, 2013)

Nesta abordagem procura-se formular uma visão geral do que se pretende desenvolver, identificando os vários subsistemas, inicialmente sem grande detalhe, cada subsistema é depois refinado em sucessivos subsistemas até reduzir os mesmos aos seus elementos base. Esta abordagem está mais associada à forma tradicional de desenvolver *Software* (*Waterfall*). (Crespi et al. 2008)

2.3.2. Uma abordagem Bottom-Up

Podemos definir esta abordagem como sendo de ***síntese***. Ou seja "a operação pela qual se reúnem os corpos simples para formar compostos, ou os compostos para formar outros de composição ainda mais complexa". (Priberam, 2013)

Nesta abordagem, o *Software* é desenvolvido através da especificação detalhada dos elementos base de um sistema. Posteriormente, estes elementos base são criados e vão sendo "acoplados" de forma a constituírem um subsistema. Vários subsistemas são por sua vez acoplados, até o sistema estar completo. Esta abordagem está mais associada aos modelos que têm por base o desenvolvimento iterativo e incremental (prototipagem). (Crespi et al. 2008)

2.3.3. Formas de convergência de ambas abordagens

No Sentido de existir talvez uma maior harmonização no processo de desenvolvimento, independentemente do modelo usado, podemos constatar a existência diferentes níveis de convergência entre as duas abordagens. (Bernstein 1996)

No *Bottom-Up*, procura-se chegar à definição do *Software*, através de uma definição prévia de todos os seus componentes.

Mas, para isso, é pelo menos necessário termos uma visão global, mesmo que incompleta ao nível do detalhe do que pretendemos fazer **(Visão *Top-Down*)**. Para tal será necessário fazer algum tipo de levantamento de requisitos de forma a conseguirmos ir adicionando algum detalhe à visão inicial.

Por exemplo, com o recurso a "casos de uso", vão-se desenvolvendo diferentes funções de forma incremental, num sistema funcional, onde se vai satisfazendo desta forma, aos "poucos", todos os requisitos do sistema, até se obter um sistema completo.(Zwiers et al. 1995)

No *Top-Down*, pretende-se inicialmente conseguir definir com clareza com o cliente, o âmbito do projeto, as capacidades e requisitos do sistema. Para tal é criada documentação mais ou menos detalhado, onde estejam definidos os requisitos, de forma a conseguir controlar a sua execução e registrar quaisquer mudanças feitas ao longo de todo o processo.

Mas, a melhor forma de mitigar o risco de desenvolver algo que possa estar assente em requisitos que podem não estar corretos, é através do uso de protótipos "parciais", das "funções" onde possa ainda existir algumas dúvidas **(Visão *Bottom-Up*)**. Conseguimos através destes, validar requisitos ou identificar requisitos "escondidos", de forma a ajudar / conduzir o cliente e a equipa de desenvolvimento a chegarem mais facilmente a um entendimento, do que na "realidade" devem ser as capacidades/requisitos do *Software*.(Zwiers et al. 1995)

2.3.4. Desenvolvimento vertical e horizontal

De alguma forma, associados aos conceitos de *Top-Down* e *Bottom-Up*, estão dois outros conceitos que lhes são por isso similares.

São os conceitos de desenvolvimento horizontal e vertical. Ambos os conceitos podem ser utilizados no *Top-Down* e *Bottom-Up*, apesar de existir uma prevalência para:

- O desenvolvimento vertical no *Bottom-Up*, que está mais relacionado com o desenvolvimento Ágil de *Software.*
- O desenvolvimento horizontal no *Top-Down*, que está mais relacionado com o desenvolvimento tradicional de *Software.*

Um bolo (por exemplo de aniversario) ou até um elefante, só se consegue comer um bocado de cada vez. Não sendo a favor da violência contra os animais ou de comer carne crua, vamos por o elefante metafórico de lado, e fazer uma analogia com o bolo no sentido de perceber as diferenças entre o desenvolvimento horizontal e vertical.

Não podemos comer um bolo grande por inteiro, como não conseguimos resolver um problema complicado de uma vez só. Nesse sentido procuramos dividir o bolo ou o problema em bocados mais fáceis de por à boca ou resolver. Mas qual a melhor maneira de partirmos um bolo ou resolvermos um problema?

Parece-me que uma parte da solução para o desenvolvimento de *Software* esta há muito tempo a olhar-nos de frente, e nós com a gulodice nem nos apercebemos disso. Partimos instintivamente um bolo por fatias verticais.

Ilustração 3 - Fatia e bolo de Chocolate[9]

Poderíamos teoricamente parti-lo na horizontal, por camadas. Na foto apresentada, podemos ver que este bolo durante a fase de construção, foi de facto cortado ou construído por camadas horizontais (para colocar uma camada de creme no meio e cobertura).

Independentemente de como se corte o bolo, este vai acabar sempre por ser comido. Mas uma coisa é certa, construir o bolo horizontalmente faz sem dúvida mais sentido. Construi-lo verticalmente (fatia a fatia) seria teoricamente possível, apesar de ser potencialmente mais trabalhoso, e o resultado final poder ser esteticamente menos apelativo.

[9] Http://marthastewart.lionbrand.com/patterns/L10637.html

No caso do *Software* ou até de uma casa, a questão já é bem diferente, se começarmos a construir uma casa, de baixo para cima, por camadas, e a meio da obra tivermos um problema que nos impeça de terminar a obra, não vamos conseguir tirar proveito do que já foi feito, não podemos dormir na casa, pois esta não tem telhado. No caso do *Software* desenvolvido horizontalmente o resultado acaba por ser parecido ao da casa, este é normalmente desenvolvido por camadas como uma casa de baixo para cima, normalmente distinguem-se 3 ou 4 camadas, (as quais ainda poderão se divididas em subcamadas):

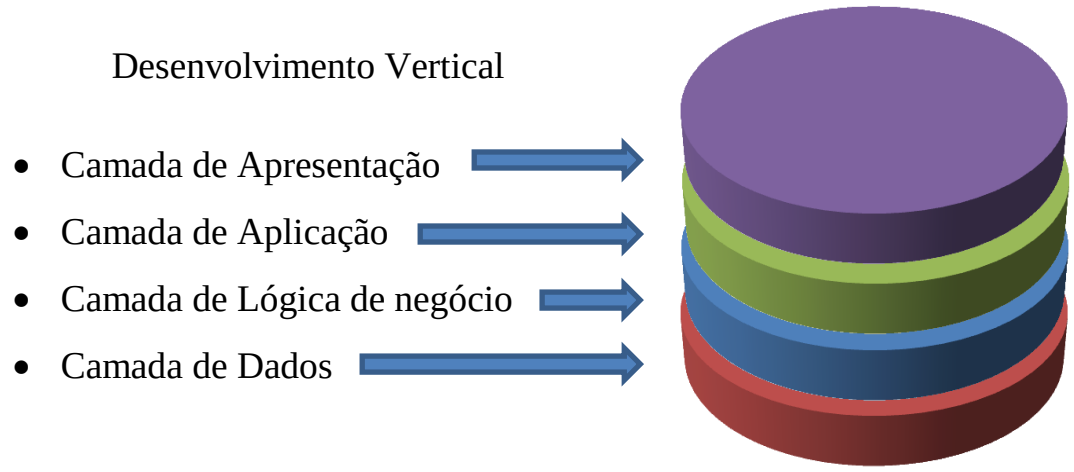


Figura 14 - Desenvolvimento Horizontal

Representação de Bolo cortado em "Camadas"

Se a meio do desenvolvimento por camadas horizontais, por alguma razão tivermos de parar o desenvolvimento (por desentendimentos entre envolvidos, falta de dinheiro, etc.). Não existe qualquer valor criado para o cliente, apesar de todo o trabalho efetivamente feito. Tanto o cliente como quem desenvolve ficam a perder porque ninguém consegue retirar valor do que foi feito.

Caso tivesse optado pelo desenvolvimento vertical, e recorrendo novamente à analogia com a construção de uma casa feita desta forma, todos os passo da construção feita habitualmente na horizontal são feitos neste caso na vertical.

Neste caso, construiríamos a casa por módulos, divisão a divisão. Poderíamos já ter construído um quarto, uma casa de banho, antes de o projeto ser interrompido. Poderíamos Hipoteticamente habitar as divisões já construídas, pois estes módulos seriam totalmente independentes, construídos desde as fundações até ao telhado.

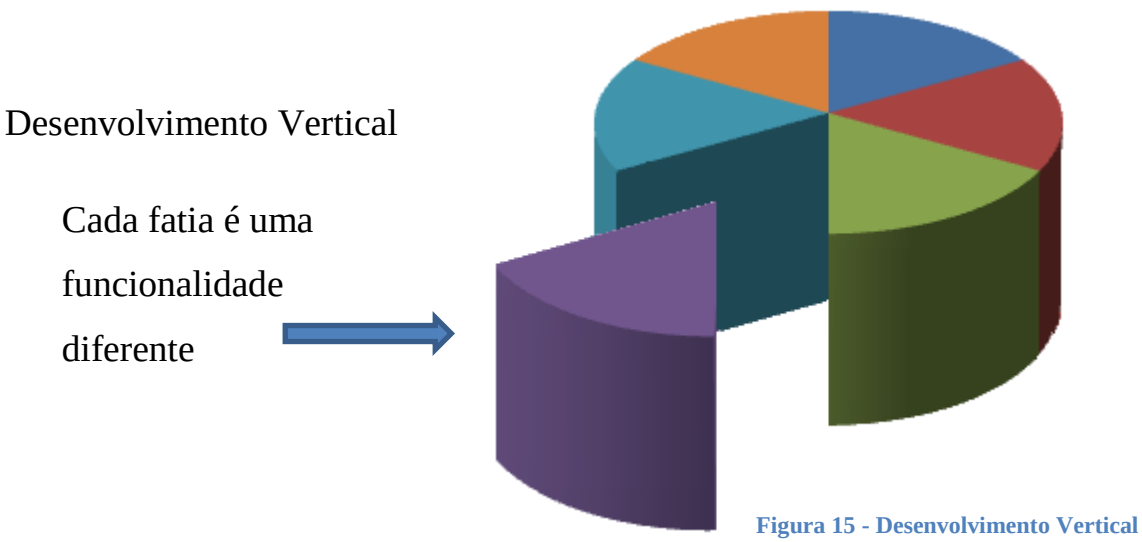


Figura 15 - Desenvolvimento Vertical

Representação de Bolo cortado em "Fatias"

No caso do *Software*, cada fatia representa uma "funcionalidade" do sistema, a escolha da ordem em que estas devem de ser criadas, deve de ser feita normalmente em função do que representam maior valor para o cliente. As funções devem de estar completamente funcionais (quando possível).

Desta forma em qualquer altura, em que o projeto tenha de ser terminado, o cliente pode tirar proveito da porção já concluída do projeto. Por outro lado renegociar com o cliente as condições do projeto será certamente mais fácil, quando este já viu, ou até já pode estar a usar parte da solução.

Ao inverso do que acontece com o desenvolvimento feito horizontalmente, o cliente dificilmente conseguirá ver a luz ao fundo do túnel, quando lhe tentarmos explicarmos que precisamos de mais uns meses ou de mais recursos, para concluir o projeto visto estarmos já a terminar o desenvolvimento da camada de lógica de negócio. Uma visão que certamente será tudo menos logica para o cliente.

3. Fatores que influenciam o Sucesso do Desenvolvimento de Software

O Standish Group, no seu relatório de 2013 (relativo a dados de 2012), em que foram analisados dados referentes a cerca de 50000 projetos recolhidos em todo o mundo, dos quais, 60% são dos Estados Unidos, 25% são da Europa e 15% são de outros países, definiu uma lista dos 10 fatores principais, que influenciam o sucesso no desenvolvimento de *Software (em projetos pequenos)*. A lista seguinte é apresentada por ordem decrescente de importância, tendo sido atribuída uma pontuação a cada um dos fatores de forma a medir o grau de importância de cada um, de um total de 100 pontos: (Standish Group 2013)

Apoios da Gestão executiva – Foram atribuídos 20 pontos a este fator. O apoio de quem patrocina o projeto é essencial. Neste contexto é muito importante a cultura organizacional existente, tanto da parte de quem manda criar o *Software* como do lado de quem desenvolve (quando se recorre a uma organização externa para executar o desenvolvimento).

Wiegers ao parafrasear Larry Constantine, afirma que,

"O que destinge "um conjunto de tolos a viajar num autocarro" (provavelmente sem destino!) de "uma boa equipa", é o facto desta ultima ter uma cultura compartilhada (sabem para onde vão)"

O sucesso de um modelo, passa sem dúvida pela forma como este é implementado, de forma a conseguir tirar melhor proveito das vantagens e melhoria dos processos que este possa preconizar. Mas, talvez um dos fatores mais importantes para o sucesso no uso de um modelo possa está mais diretamente relacionado com a cultura organizacional existente nas organizações envolvidas. Ou seja, a partilha de valores e princípios que possam guiar os comportamentos, as atividades, as prioridades e as decisões de um grupo de pessoas que trabalham numa mesma área. A existência de crenças comuns entre colegas. (Wiegers 1996)

Qualquer adoção de um modelo requer uma aprendizagem lenta, uma adaptação ao contexto da organização (das pessoas e equipas) onde este vai ser usado e aos tipos de projetos que se possa querer desenvolver com o mesmo. A cultura organizacional tem aqui um papel relevante, no sentido de ser um grande catalisador da mudança.

Quando esta cultura permite fazer experiencias, e aceita que estas possam fracassar, quando não se critica por errar, mas se procura saber que erros foram cometidos, e o que podemos apreender com estes, o que podemos fazer de diferente no sentido de poder melhorar o resultado obtido na próxima tentativa (critica construtiva), passa a existir então espaço e condições favoráveis para a inovação.

"Quando os envolvidos percebem que o risco de insucesso é aceitável, estão mais dispostas a explorar novas formas de fazer as coisas. Se no entanto, o modelo for imposto "cegamente" pelas chefias, ou seja, sem levar em conta se este modelo se adapta bem, quer aos problemas que se quer resolver, quer ao ambiente em que este se pretende aplicar, podemos certamente esperar um mar bastante picado durante toda a viagem, isto se o navio não naufragar a meio caminho". (Wiegers 1996)

Envolvimento dos utilizadores - Foram atribuídos 15 pontos a este fator. Sabemos que o envolvimento e colaboração dos utilizadores é um dos requisitos essenciais para o sucesso de um projeto, independentemente do tamanho deste. A disponibilidade de permanência de alguém que represente os utilizadores junto da equipa de desenvolvimento é uma mais-valia. Os utilizadores devem de estar altamente motivados, e sentir que são também eles, elementos responsáveis pelo projeto.

Otimização - Foram atribuídos 15 pontos a este fator. Quanto mais pequeno o projeto, maior é a probabilidade de este ter sucesso. O Standish Group acredita que grande parte dos projetos podem ser divididos em projetos mais pequenos, Se 20% das funções e os recursos criados num *Software*, providenciam 80% do valor do criado por este, a criação de projetos mais pequenos, focados na criação de valor, podem aumentar o valor global destes últimos. Este facto pode ajudar a reduzir custos, prazos e frustrações.

Qualificados dos recursos - Foram atribuídos 13 pontos a este fator. Ter as pessoas certas, a fazer as coisas certas e quando esta têm de ser feitas, é meio caminho para o sucesso de um projeto. Neste contexto, a competência é essencial, ou seja, ter as qualificações suficientes para desempenhar as tarefas necessárias, ditadas pelos requisitos de projeto. Ter a experiencia na vida real no domínio do problema (já tenha desenvolvido software similar).É também importante ter boas capacidades de relacionamento interpessoal e boas capacidades comunicativas, no sentido de conseguir demonstrar as suas qualificações de forma a dar confiança aos outros participantes do projeto. Outros pontos essenciais são por exemplo, a motivação, a formação continua, o espírito de equipa e a existência de uma boa química entre todos os envolvidos.

Experiencia na gestão de projetos - Foram atribuídos 12 pontos a este fator. A existência de um gestor de projetos competente. Que tenha a capacidade de tomar as decisões corretas, no momento correto e consiga descomplicar um processo complicado, definindo um plano, de forma a tornar o mesmo executável.

Processos Ágeis - Foram atribuídos 10 pontos a este fator. A filosofia associada aos modelos ágeis, os valores e princípios defendidos por estes já demonstraram que funcionam, para além disso estes processos Ágeis definem diretrizes gerais para o desenvolvimento de *Software*, as quais seguem todas as melhores práticas atuais. Esta filosofia encoraja a satisfação do cliente, através participação e comunicação ativa e continua entre todos os envolvidos. Procura fazer uma entrega incremental de *Software* Funcional, recorre para isso, a equipas pequenas de desenvolvimento altamente motivadas.

Definição de objetivos de negócio claros - Foram atribuídos 6 pontos a este fator. A existência de claridade em relação aos objetivos que se pretende alcançar, é essencial para o sucesso de um projeto. Esta claridade facilita que todos os envolvidos se foquem nos objetivos. Cada Stakeholders tem a sua agenda própria que pode ter objetivos diferentes dos objetivos gerais, quanto maior o projeto, mais agendas existem para gerir. A definição clara dos objetivos e a atribuição de prioridades e valor a cada um destes ajuda a que não haja um afastamento do caminho certo.

Maturidade emocional - Foram atribuídos 5 pontos a este fator. Desenvolver *Software* é um processo social, desenvolvido em equipa, com a colaboração de pessoas internas e externas a uma organização. Ter maturidade emocional é ter a capacidade de perceber o mundo que nos rodeia, perceber onde nos enquadramos, a forma como reagimos perante os outros e a forma como conseguimos gerir as nossas relações.

Execução - Foram atribuídos 3 pontos a este fator. A capacidade de **execução** é a de conseguir fazer o barco chegar a bom porto. Para tal é essencial a existência de uma definição do problema, de um bom plano para o solucionar e de regras que possam ajudar a que se consiga obter sucesso, no sentido de que se consiga corrigir mais facilmente qualquer desvio em relação ao que está definido.

Ferramentas e infraestruturas - Foram atribuídos 1 ponto a este fator. As ferramentas e infraestruturas existentes devem de proporcionar uma ajuda na concretização dos projetos. No entanto muitas vezes as organizações ficam dependentes (reféns) destas ferramentas e infraestruturas (escolhidas anteriormente) em vez de conseguir escolher o que faz mais sentido para um determinado projeto. No entanto a estandardização, do uso destas, ajuda a que os envolvidos ganhem maior competência na sua utilização, ficando mais familiarizados com o tempo e experiencia adquiridas com a sua utilização.

Podemos constatar que os fatores apresentados estão na sua maioria relacionada com fatores humanos ou organizacionais.

4. Conclusões: a importância das pessoas e da comunicação

Através da analise bibliográfica feita, fiquei com a clara perceção de que ao longo dos anos temos felizmente vindo a assistir a mudanças no desenvolvimento de *Software*, que de facto acabam por, se bem que tardiamente, quase 45 anos depois, dar razão ao que Royce realmente defendia no seu artigo. Mais vale tarde, do que nunca.

Royce foi de facto um visionário, se fosse no tempo da inquisição talvez o tivessem considerado um herege, como aconteceu com Galileu, quando este veio provar as afirmações de Copérnico, de que a terra não estava no centro do universo, ideia tão contrarias à doutrina de então da igreja, que blasfémia! Galileu safou-se de ir parar à fogueira, porque concordou (foi pressionado!) em negar as afirmações feitas anteriormente. Teve sorte em lhe permitirem a negação, muitos não tiveram a mesma sorte.

Outra curiosidade que demonstram que as coisas não são sempre o que parecem, é a definição do dicionário da palavra Maquiavélico (Priberam 2014):

(Nicolau *Maquiavel*, antropónimo + *-ico*)
Adjetivo

1. Do maquiavelismo ou a ele relativo.
2. Em que predomina a astúcia, a má-fé e o oportunismo.
3. Pérfido, ardiloso, velhaco.

Mas porque associar todos estes adjetivos pouco abonatórios à pessoa de Maquiavel?

Talvez porque este defendeu a ideia, pouco racional, pelo menos à época, de que quem devia de estar no poder, não deveria de estar lá por imposição divina, mas sim pelo seu mérito. Defendia a ideia de que deveriam ser as pessoas a decidir… pouco sensato ir contra o poder instituído na altura (monarquias e clero). Por isso teve o direito, não de ir parar à fogueira, mas de ser mal adjetivado, para toda a eternidade. De certa forma criou-se um pretexto para que as pessoas não fossem ler os seus ensinamentos.

Muita de nós, pelo menos as pessoas ligadas as TI, já ouviu falar de Alan Turing, mas poucos conhecem a tão triste história por detrás da lenda que foi este homem. Conhecemos Alan Turing, pelo seu papel preponderante, na descoberta da forma de decifrar as mensagens dos Alemães, durante a 2ª guerra mundial, dando desta forma um grande contributo para que esta terminasse muito mais cedo. Ajudando a poupar desta forma um número incalculável de vidas.

É por muitos considerado também o pai da ciência computacional e da inteligência artificial pelos contributos dados por ele. Entre outras coisas, Alan Turing foi quem formalizou os conceitos de algoritmo e computação.

No entanto pouco sabem que após todo o seu contributo dado, o governo Inglês conseguiu condena-lo em 1952, pasme-se, por "indecência grosseira", pelo facto de este ser homossexual, prática sem dúvida abominável, e que na altura ainda era criminalizável.

Turing aceitou (foi forcado como o Galileu) a submeter-se a tratamentos com hormonais femininas (castração química), em alternativa à prisão. Talvez em função de tudo o que lhe aconteceu, este suicidou-se em 1954 poucos dias antes de completar 42 anos! Não gosto nem de pensar no contributo extra que Turing deixou de dar por tão sinistra razão.

Pelo menos também a Turing, se bem que tardiamente, neste caso, a Rainha de Inglaterra, a poucos dias do natal de 2013, acabou por "perdoar postumamente" os "pecados" cometidos por este. Talvez uma prenda para a comunidade homossexual do seu Reino, visto a proximidade do natal.

Isto leva a pensar que talvez hipoteticamente falando, que se Pilates desse a escolher aos Judeus condenar Turing (Jesus) ou Hitler (Barabbas), os Judeus escolheriam salvar Hitler! Passamos a vida a fazer más escolhas…

Mas em Portugal também tivemos o nosso blasfemo, basta lembrar o que tentaram fazer cá na nossa Pais, com o Saramago, e que de certa forma até conseguiram, não fosse o que foi feito atenuado pelo premio Nobel atribuído a Saramago (malditos Suecos). Mesmo assim, este "sentiu" a necessidade de virar costas ao pais que o viu nascer, como

forma de protesto, mandando dessa forma "os governantes da altura para as ortigas", tendo optado por se auto exilar na ilha de Lanzarote.

Mas talvez a pergunta mais importante que se deveria fazer neste caso era:

".Quantos dos que o criticaram, leram efetivamente o "O Evangelho segundo Jesus Cristo"?

Se calhar contam-se pelos dedos de uma mão de um maneta!"

Se calhar foram tantos como os que leram o Artigo de Royce…

Num *Email* trocado há dias com Alister Cockburn, um dos signatários originais do "Manifesto Agile", apresentei-lhe algumas das conclusões a que cheguei neste trabalho, e em relação a Royce, este afirmou o seguinte:

"O artigo de Royce é um choque e tanto – poucas pessoas sabem o que ele realmente disse, e até aqueles de nós que já lemos múltiplas vezes o artigo, continuamos a argumentar sobre o seu significado.

É um artigo profundo e rico, especialmente, aplicando o mesmo ao mundo da programação de hoje, que é muito diferente (de à mais de 40 anos Atrás) "

De facto o artigo de Royce pode ser considerado um serio candidato ao premio de "um dos mais mal interpretados artigos de sempre", e com consequências tão dramáticos que ainda hoje se fazem sentir os seus efeitos devido à adoção cega do modelo *Waterfall*. Para perceber esse impacto, basta primeiro pensar no que se conseguiu fazer ao longo destes últimos 40 anos, graças as TI.

Cientes desta mais-valia obtida, bastava agora fazermos o cálculo (decerto muito complicado e dificilmente representável), de todas as horas/homem perdidas em todos os projetos falhados, cuja causa pode estar de alguma forma associada ao uso do *Waterfall*. Conseguiríamos ter uma noção, se bem que aproximada, do que

potencialmente se poderia ter produzido a mais, ao longo destes últimos 40 anos. Talvez, o Homem tivesse já conseguido chegar a Marte (missão tripulada).

Coincidência ou não assistiu-se nos últimos 10 a 15 anos, a um crescimento exponencial do uso das TI. Certamente existirá alguma relação entre este crescimento, e a adoção progressiva de novos modelos mais ágeis, para além de existir uma influencia positiva das correntes de pensamento associadas a estes modelos, não só no desenvolvimento de *Software*, mas também nas áreas da:

- Comunicação
- Cultura organizacional
- Gestão de projetos

Analisando todas as mudanças feitas ao longo de décadas na forma como se tem vindo a desenvolver *Software*, podemos constatar que estas mudanças podem até ter diferentes objetivos. Mas na realidade existe um denominador comum entre todos elas, que é o facto de através dos modelos procurar-se sempre obter o melhor resultado final possível no desenvolvimento de *Software*, o que nem sempre se consegue. Mas isto, pelo menos em grande parte, quanto a mim, não é culpa do modelo.

Jeff de Luca afirma que "as TI são 80% psicologia, e 20% de tecnologia". Talvez seria mais certo dizer que as TI são 80% de um misto de psicologia e sociologia, visto que o sucesso destas não dependerem apenas de questões do foro interno de cada individuo, mas talvez, e acima de tudo, na forma como todos os indivíduos envolvidos se relacionam e **comunicam**. E de facto podemos constatar que as propostas feitas em modelos mais recentes têm vindo a dar cada vez maior enfase ao fator que hoje sabemos ser o mais importante, e que tem maior **impacto** no sucesso das TI, que são as **pessoas**.

Mas mesmo no desenvolvimento tradicional *Waterfall*, em que se procura criar um plano, muita documentação, etc. O objetivo não é só o de melhorar o resultado final mas também, e sobretudo, procurar arranjar formas de lidar com questões de complexidade, de forma a facilitar a vida a quem implementar o modelo, que são as **pessoas**.

É frequente lermos ou ouvirmos, críticas a cerca deste modelo por causa da sua "pseudo rigidez", por supostamente lhe faltarem conceitos como o da iteração ou prototipagem.

Mas Royce de facto afirma categoricamente no mesmo artigo que consagra, de certa forma erradamente, a visão que muitos têm do *Waterfall*, de que em teoria este parece ser excelente, porém… na realidade …raramente resulta. Para além de defender também no mesmo artigo, o conceito de "*Do It Twice*" onde este enfatiza a necessidade de se fazer pelo menos uma ronda (iteração) de **prototipagem**, à qual Royce dá o nome de simulação. O objetivo é o de se conseguir garantir que questões mais problemáticas, algumas das quais certamente estarão relacionadas com problemas de **comunicação** entre as **pessoas**, sejam detetadas na primeira iteração, e possam ser posteriormente compreendidas e corrigidas, numa segunda iteração, porque as **pessoas** tendem a aprender com os erros.(Royce 1970)

O facto de se procurar desenvolver de forma incremental, demonstra a consciência de que desta forma se consegue desconstruir um problema que à partida possa parecer de difícil compreensão ou resolução, em vários problemas mais facilmente percetíveis e solucionáveis (fator psicológico). A **comunicação** entre os intervenientes torna-se mais fácil, com a segmentação do problema, em fatias mais finas e simples. Este fator é essencial para ajudar na capacitação (acreditar que se consegue), das **pessoas** envolvidas (equipa de desenvolvimento, cliente, utilizadores, etc.) de que o projeto irá chegará a bom porto, o que é meio caminho andado para que se consiga realmente ter sucesso.

O que o modelo em Espiral vem trazer de novo, é o facto de este procurar ativamente identificar, analisar e reduzir riscos no processo de desenvolvimento, através do uso de **protótipos** numa fase inicial do processo, no sentido de conseguir validar o mais precocemente possíveis requisitos não identificados inicialmente ou em que existe alguma incerteza quanto aos mesmos. O recurso à **prototipagem,** neste modelo, surge como forma de validação dos requisitos, no sentido de se tentar reduzir o fosso entre as visões dos intervenientes, que muitas vezes não são coincidentes. Demonstra-se assim que (grande) parte dos riscos possa existir em função de problemas de **comunicação** e perceção entre as **pessoas** envolvidas.

No desenvolvimento rápido de aplicações (RAD), a enfase principal, esta em conseguir-satisfazer mais rapidamente as necessidades do negócio (do cliente). Surge o conceito de priorização no desenvolvimento das várias "funções" a integrar na aplicação, em

função do valor que casa uma tem para o cliente. Este conceito, tem atualmente um valor de destaque em todos os modelos mais recentes.

Nesse sentido torna-se imperativo o envolvimento "ativo" do cliente/utilizador ao longo de todo o processo, no sentido de conseguir estabelecer maiores sinergias (consensos) entre todos os envolvidos, ou seja melhorar a **comunicação** entre estes, nomeadamente com recurso a *Workshops* (JAD - *Joint Application Development*).

A equipa de desenvolvimento, através do uso ferramentas de desenvolvimento rápido de aplicações (geradoras de código, interface de utilizador, etc.), fica mais liberta para pensar nas funções de negócio a ser desenvolvidas, pois não tem de se focar tanto na canalização, ou seja, por exemplo a analise e implementação das tecnologias a ser usadas (camadas inferiores). A prioridade é assim dada as **pessoas**. Existe um uso mais efetivo da **prototipagem**, visto esta estar associada normalmente a ferramentas de desenvolvimento rápido de aplicações, o que permite que iterativamente se vá produzindo **protótipos** evolutivos e "entregáveis". É também aplicado o conceito de *Timeboxing*, de forma a evitar atrasos nos prazos de entrega, preferindo-se neste caso reduzir nos requisitos entregues, do que prolongar os prazos de entrega.

O desenvolvimento Vertical, que está de certa forma ligado à abordagem *Bottom-Up*, está intrinsecamente ligado à questão da necessidade de criar desde o início do desenvolvimento o maior valor possível para o cliente, para que, caso o projeto tenha por alguma razão de ser terminado, não exista uma perda total, quer do trabalho feito até ao momento pela equipa de desenvolvimento, quer do investimento até então realizado pelo cliente.

Se fizermos uma analogia com a construção de uma casa, no desenvolvimento horizontal, que está de certa forma ligado à abordagem *Top-Down*, seria necessário fazer o projeto, depois começa a fase de construção. Esta é iniciada pelas fundações, depois constrói-se os pilares, as paredes… etc. Se por acaso o projeto tem de ser terminado, por exemplo por falta de fundos. O dono da obra não pode tirar nenhum proveito do trabalho já feito, nem teoricamente, nem na prática, pois não pode habitar a casa sem esta ter, por exemplo um telhado.

Já ao contrário, com o desenvolvimento vertical, pelo menos em teoria, e até de certa forma na prática, desde que não aparecesse o fiscal da câmara, a proibir a utilização da parte da casa já construída, por falta da licença de habitação, seria possível habitar as divisões da casa já construídas, porque estas tinham sido construídas de forma modelar, divisão a divisão, cada uma desde as fundações até ao telhado.

Podemos alegar que seria mais fácil assentar o telhado todo de uma vez, e de facto não deixa de ser verdade. Mas até lá não poderia tirar nenhum proveito de todo o trabalho até então feito.

Procura-se em todos os modelos, dar cada vez maior enfase aos fatores que hoje sabemos serem os mais importantes, e que têm maior impacto no sucesso das TI, que são os que estão de alguma forma relacionados com as pessoas (Cockburn & Highsmith 2001), como:

- A cordialidade, forma como todos os intervenientes se relacionam.

- O desenvolvimento de competências, tanto a nível individual, como a nível colaborativo
- A habilidade ou talento, conseguir por em prática as técnicas e conhecimentos adquiridos
- A capacidade de comunicação, a forma como a informação é recebida e enviada.

Destes fatores, do meu ponto de vista, talvez o mais importante, e também em muitos casos o mais subestimado, é o da comunicação. No entanto até neste caso, a evolução dos modelos de desenvolvimento vem tentar colmatar este problema. De facto a fixação inicial existente nos modelos mais tradicionais, sobre a necessidade de se produzir boa documentação, por vezes demasiado “extensiva”, tem vindo a ser posta de lado, pelos modelos mais recentes em favor de outras abordagens. Mas não se pense que isto se deve simplesmente ao facto de por exemplo, se poder considerar que esta prática é aborrecida, ou que esta apenas traz mais complicações, à tarefa já de si difícil de desenvolver *Software*. Quantidade pode não significar qualidade.

Estudos realizados na área da "teoria da riqueza de informação", realizados por exemplo por Richard L. Daft and Robert H. Lengel, apontam para o facto, de que de todos os meios de comunicação, a escrita é o canal menos rico e eficaz. (Cockburn 1999)

Os mesmos estudos revelam também, conforme podemos constatar na figura 5, que de todos os meios, o mais rico e eficaz é a comunicação "cara á cara". E que este meio ainda pode ser melhorado, por exemplo, com o uso de um simples "quadro branco". Desta forma os intervenientes podem recorrer ao quadro para expressar graficamente, o que possam não esta a conseguir transmitir "cara á cara". Este é uma forma extra de validação, uma forma de prototipagem de baixo nível.

E porque na realidade…

"Uma imagem pode ajudar a definir mais de mil palavras"

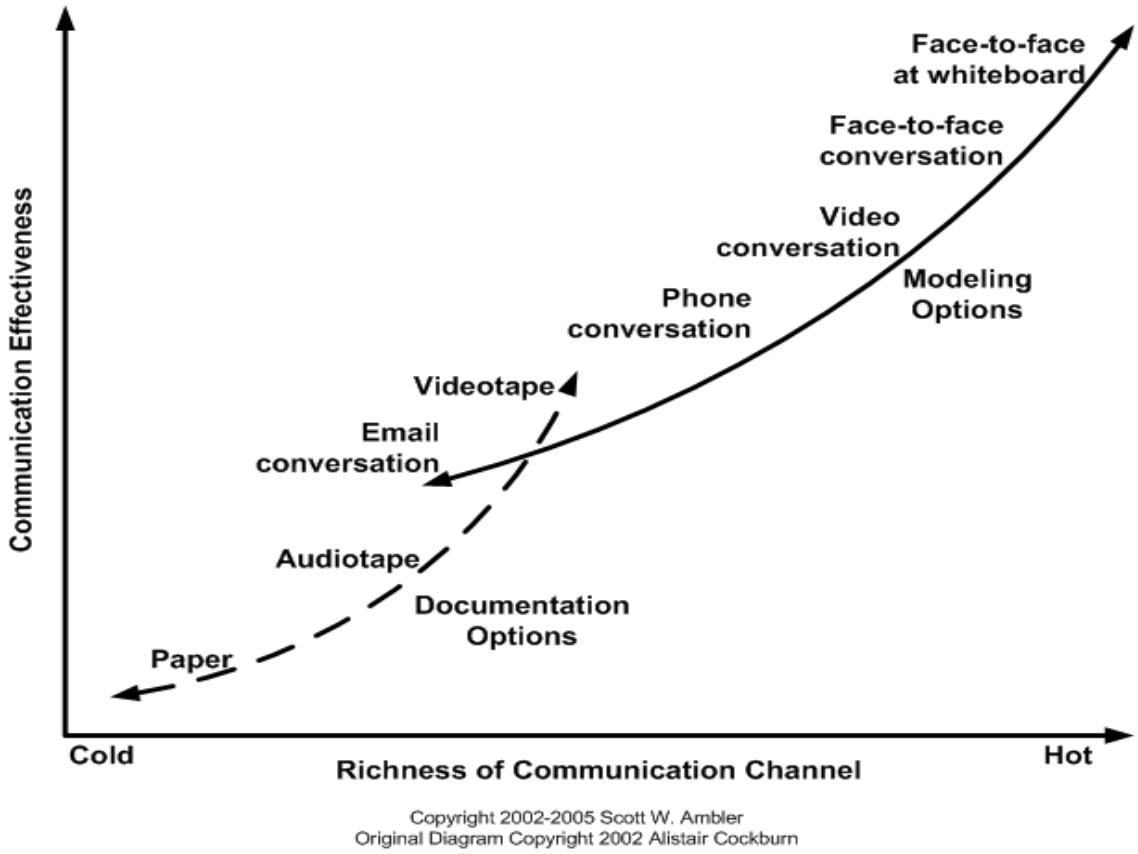


Figura 16 - Representação dos vários meios de comunicação: eficácia e riqueza dos mesmos(Cockburn 1999)

Nos modelos mais recentes constatamos que não se procura eliminar a documentação, só por eliminar, mas sim reduzir a mesma na sua essência ao que realmente importa documentar, de preferência depois de a informação nela contida ter sido obtida e validada através de meios de comunicação mais eficazes.

Conforme se pode verificar na figura 5, à medida que nos vamos afastando do nosso interlocutor existe uma perda de eficácia, perde-se a capacidade de se comunicar através de técnicas não-verbais, como gestos e expressões faciais. A capacidade de alterar o sincronismo e inflexão vocal também é perdida, as pessoas não comunicam apenas através das palavras que dizem, mas também pela forma como dizem essas mesmas palavras.

Existe por exemplo a possibilidade de darmos mais enfase a esta, ou aquela parte de uma conversa. O facto de conseguirmos obter e dar feedback instantâneo, permite-nos perceber de uma melhor forma, se o nosso interlocutor percebeu o que estávamos a querer transmitir e vise versa.(Cockburn 1999)

Mas o facto de podermos recorrer a um meio mais rico e eficaz, não é por si só garantia de que a informação assim obtida esteja totalmente correta, isto porque não estamos a ter em consideração possíveis erros de comunicação e a existência quase certa de falhas de perceção, possíveis mesmo com o recurso a um meio rico.

Mas, o certo é que as discrepâncias que possam ainda persistir entre os envolvidos, quando for usado um meio rico estarão muito mais minimizadas do que quando se recorre ao uso de um meio mais pobre.

Na análise bibliográfica, realizada, ao tema do desenvolvimento de *Software*, foi me difícil encontrar referências diretas, em relação aos problemas de “comunicação” e “perceção”, que não tivessem uma abordagem apenas tecnológica, não é que não existam, são disso exemplo, alguns artigos de Alistair Cockburn ou do livro *Communicating Project Management* de Hal Mooz, Kevin Forsberg e Howard Cotterman. Mas, é muito mais comum encontrarmos, artigos que propõem soluções técnicas para problemas que não o são.

5. Considerações Finais e Trabalhos futuros

5.1. Considerações Finais

No final da análise feita neste trabalho, podemos então concluir que o maior problema que assola já há décadas o desenvolvimento de *Software*, não é o aumento de complexidade, nem da incerteza, nem se deve na essência ao uso de modelos ou metodologias de desenvolvimento erradas. Isto porque novos modelos certamente ajudam pelo menos na solução de 20% dos problemas existentes anteriormente. O problema é mais transversal do que se possa pensar. Acima de tudo estamos perante problemas de falhas de comunicação e erros de perceção entre envolvidos. De facto o que todos os modelos de desenvolvimento ao longo dos anos têm tentado fazer, e diga-se de passagem com algum êxito, é conseguir mitigar os problemas resultantes da complexidade associada à comunicação.

Fazendo uma analogia com as redes de computadores, o ideal seria que conseguíssemos não só eliminar todas as perdas e corrupção de dados entre emissor e recetor, mas também que o recetor conseguisse transformar esses dados em informação valida, ou seja que este conseguisse descodificar corretamente os dados enviados pelo emissor.

O problema é que as chaves de cifrar/decifrar usadas nem sempre são as mesmas. Neste caso o pior é que podemos nem sequer nos aperceber que a chave usada para decifrar não era a correta. Recebemos informação que até parece ser valida e a percebemos como tal, mas está na realidade é diferente da que o emissor pensa termos recebido. Não podemos usar um *Checksum* para saber se o que alguém nos transmitiu foi por nós interpretado da mesma forma. São variáveis a mais para conseguirmos resolver com um algoritmo ou um protocolo.

O que se pensa dizer,

não é o que se diz,

e o que se diz,

não é captado na totalidade,

o que é captado é diferente do que é interpretado,

…Esta interpretação é certamente muito diferente,

do que foi inicialmente pensado.

Se pensarmos que varias mensagens simples podem querer dizer o mesmo que uma mensagem mais complicada. É no entanto mais provável percebermos um conjunto de mensagens simples do que uma mensagem complicada. E se nos repetirem algumas vezes estas mensagens simples, conseguiremos eventualmente perceber a mensagem mais complicada. (conceitos iterativo e incremental)

Podemos também fazer uma analogia com as redes de computadores quanto à importância da prototipagem no desenvolvimento de *Software*, no sentido em que esta nos permite detetar e corrigir de forma eficaz erros na transmissão de informação entre emissor e recetor, mesmo quando se utiliza canais (meios) de transmissão que todos sabemos ou deveríamos de saber não ser fiáveis. Este facto aplica-se tanto à comunicação, como às redes de computadores. Através do uso de protótipos conseguimos transpor os requisitos de um documento de interpretação complicada, para uma realidade muito mais tangível. Conseguimos validar ou corrigir mais precocemente os requisitos com o feedback também ele mais precoce dos Stakeholders.

Estes ajudam a eliminar ou reduzir o fosso que possa existir entre a visão que o cliente tem do produto, e o entendimento que a equipa de desenvolvimento possa ter do mesmo. Consegue-se chegar mais rapidamente a um entendimento comum do que devem ser os requisitos do sistema, o que ajuda a reduzir o risco potencial de insatisfação do cliente em relação ao produto final.

Cockburn faz as seguintes perguntas, acerca da Cooperação e comunicação (Cockburn 2004), que…

"Se nos perguntarmos (a nós próprios), o que é o desenvolvimento de *Software*?

O que podemos descobrir?"

No meu caso, eu acabei por descobri as respostas, de forma tardia, através de um longo processo pelo qual passei para desenvolver este trabalho. Foi-me inicialmente difícil de encontrar um tema para o trabalho, passei por vários. Fui substituindo um tema por outro, porque cada vez que aprofundava um tema, tropeçava noutra abordagem que para mim me parecia mais interessante, até que cheguei a este tema. O qual espero ser final, mas que certamente não será o ultimo. E após toda a análise feita acabei por concordar com a afirmação de Cockburn,

"Que o desenvolvimento de Software é composto na realidade por pessoas, que inventam e comunicam entre si." (Cockburn 1999)

Isto porque segundo Cockburn, estas pessoas:

- Procuram resolver um problema que não compreendem inteiramente, e que está em constante mudança, mesmo de baixo dos seus narizes.
- Criam soluções que não compreendem inteiramente, as quais estão em constante mudança, mesmo de baixo dos seus narizes.
- Procuram expressar os seus pensamentos em linguagens artificiais que não compreendem inteiramente, as quais estão em constante mudança, mesmo por baixo dos seus narizes), para a um intérprete que é impiedoso perante os erros (o computador e/ou o meio envolvente/utilizadores do software),
- Onde cada escolha tem consequências económicas, e onde os recursos são limitados.

Ou seja, o desenvolvimento de *Software* é um "jogo" cooperativo, com recursos limitados, objetivo definido, cujos movimentos possíveis consistem em:

- Inventar
- Decidir
- Comunicar

Podemos constatar, que no entanto ao longo dos anos se tem procurado essencialmente melhorar o desenvolvimento de *Software*, à custa de mudanças tecnológicas, com a criação Modelos e metodologias novas. Apesar de supostamente cerca 80% dos problemas associados ao desenvolvimento se poderem considerar da área da psicologia ou sociologia (das pessoas). Existem no entanto poucas soluções provenientes destas duas áreas que mais influencia têm no sucesso das TI (os tais 80%).

Hipoteticamente, e recorrendo ao princípio de Pareto (80/20), se procurássemos solucionar os 80% de problemas, que estão de alguma forma relacionados com as pessoas, com soluções provenientes dos campos da psicologia ou sociologia, conseguiríamos talvez resolver grande parte desses problemas, gastando apenas 20% do esforço gasto ao longo dos anos, com soluções técnicos (modelos e metodologias).

Não sei se o valor de 80% é um valor que pode ser considerado correto, talvez seja mais "politicamente" correto, porque neste tipo de questões talvez seja difícil definir um valor certo. Mas, o facto é que através da análise feita neste trabalho, constata-se que uma parte significativa das soluções propostas para de alguma forma eliminar ou pelo menos atenuar os problemas, procuram resolver questões relacionadas com as pessoas, seja no plano pessoal, no plano relacional (entre pessoas), ou por influência da cultura organizacional ou da vida em sociedade.

Pode-se concluir que quase todas as soluções são reativas em vez de ser preventivas. Recorremos por exemplo à prototipagem porque não fomos capazes de solucionar o problema na sua fonte, que é a comunicação. Desenvolvemos verticalmente, de forma a conseguir demonstrar que somos capaz de fazer o trabalho e apresentar valor o mais cedo possível, a fim de criar um clima favorável entre todos os interessados. Mas será que não se consegue criar esse clima a priori?

5.2. Trabalhos futuros

Depois da análise feita e conclusões tiradas, cheguei à conclusão de que existe um certo vazio no campo mais sociológico e psicológico da questão. Pensei muito, em como conseguir futuramente desenvolver e conseguir por de alguma forma em prática as conclusões tiradas por mim neste trabalho. Cheguei à conclusão de que poderia optar pela criação de um **"*Framework Motivacional*",** o qual à falta de melhor nome me parece bem.

O objetivo deste *Framework* seria o de, por um lado conseguir consciencializar todos os envolvidos num projeto de desenvolvimento (e não só), para o problema da comunicação. Procurar por outro lado fomentar mudanças de atitudes, princípios e valores, a nível da cultura organizacional, dos grupos existentes dentro das organizações, das equipas e dos indivíduos.

Para tal, o *Framework*, incorporaria, uma serie de atividades, diárias, semanais, mensais, etc. que ajudem a alcançar os propósitos pretendidos.

Estas atividades podem exigir a participação direta e ativa, por exemplo a participação em palestras, ou serem mais subtis e indiretas, como a da comunicação de frases de motivação (diariamente diferentes). Outro tipo de atividades interessantes poderiam ser as de colaboração com colegas com que normalmente não se convive dentro da organização, de forma a criar uma maior coesão entre todos os colaboradores da organização.

Para garantir maiores níveis de sucesso no desenvolvimento dos projetos, no aspeto externo à organização, e no sentido de criar as sinergias necessárias junto do cliente (todos os envolvidos na definição dos requisitos), devemos de procurar perceber quais os valores e princípios do nosso cliente e da sua organização (futuros utilizadores do software). De forma a conseguirmos tanto quanto for possível fazer um alinhamento destes, aos princípios e valores da equipa de desenvolvimento (e da sua organização). Para além disso devem de ser criadas uma seria de atividades, no sentido de sensibilizar todos os envolvidos, para os problemas da comunicação, colaboração e perceção.

Para que estes consigam perceber quais as atitudes mais desejáveis, no sentido de potenciar positivamente os resultados finais.

É fundamental que se consiga também fazer algumas atividades do lado do cliente com os envolvidos no projeto, desde que haja abertura por parte destes (se não tenta-se criar essa abertura!). Apesar de falarmos à partida a mesma língua (pode até não ser o caso), os hábitos, as vivencias e a linguagem (entre muitas outras coisas) de uns e de outros pode ser muito diferente (em grande parte dos casos é).

Um exemplo concreto, e que pode efetivamente constituir uma atividade a desenvolver junto de todos os envolvidos, é o de pegarmos na ideia seguinte:

Quem esta do lado do desenvolvimento, não tem dúvida de como funciona um algoritmo (alguns até podem ter), pelo contrário do lado do cliente, poucos saberão o que é um algoritmo. Isto até tem alguma lógica… Eles não precisam saber o que é um algoritmo, para isso é que contrataram a equipa de desenvolvimento!

Mas não pode haver nada de mais errado nesta afirmação, pois a causa de muitos projetos falharem passa mesmo por aqui. Pois a equipa de desenvolvimento, sabe de automação de receitas (de algoritmos), é o seu trabalho. Esta vai efetivamente criar um *Software* para automatizar a elaboração de uma receita, por exemplo no nosso caso, um *Software* para fazer ovos mexidos.

Mas quem vai explicar como se faz neste caso os ovos, quais os ingredientes, etc. é o cliente. Pois o cliente, até só percebe mesmo é de ovos mexidos. É o seu "Core Business!". O cliente está tão habituado a fazer os mesmos manualmente, que parte do pressuposto que a equipa de desenvolvimento até sabe como se faz ovos mexidos, se calhar tanto como o cliente sabe o que é um algoritmo! O cliente passa então a explicar os passos todos ao representante da equipa de desenvolvimento, até se calhar algo contrariado, porque existe algum segredo que ele está relutante em revelar, começando por explicar "tem de partir os ovos para dentro de uma taça", etc.

Terminada a explicação, o representante da equipa de desenvolvimento até achou todo o processo bastante simples, nem precisou de ver o cliente a fazer os ditos ovos, apesar de nunca ter feito uns ovos mexidos, apesar do cliente insistir que talvez seria melhor este ver o cliente fazer os ovos mexidos. Vários conjuntos de falsos pressupostos depois…Entre os quais, a simplicidade de fazer uns ovos mexidos e o facto de que as cascas não eram para aproveitar. Temos como resultado brilhante de uma comunicação sem falhas, um *Software* perfeito para fazer ovos mexidos com cascas! O que certamente não vai corresponder ao que o cliente queria que lhe fosse entregue inicialmente.

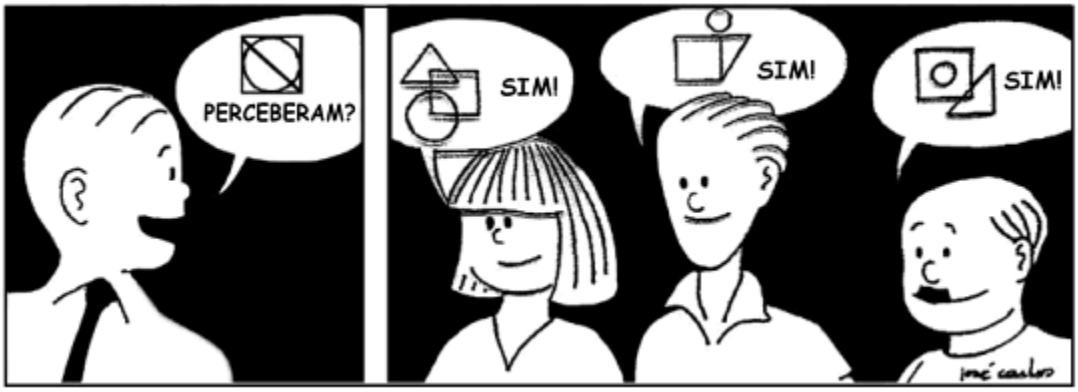


Ilustração 4 – Um problema, 3 soluções(Canario 2006)[10]

No sentido de minimizar a ocorrência de falsos pressupostos, associados à complexidade, quer seja:

- Do problema que se quer solucionar,
- Da comunicação em si
- Das próprias pessoas que recorrem à comunicação de forma pouco eficiente ou eficaz (por vezes).

[10]Disponível em http://books.google.pt/books?id=A_bOsX9LhaUC&pg=PA43&hl=pt-PT&source=gbs_selected_pages&cad=3#v=onepage&q&f=false

Ou até mais concretamente do conjunto destas 3 razões (pode haver mais), devemos procurar ativamente, que todos os envolvidos fiquem mais capacitados, através do desenvolvimento de competências a nível da comunicação (e não só!), tanto a nível individual, como a nível colaborativo, tais como:

- Ter moral, ser ético em tudo o que se faz
- Ter Confiança, em si e nos outros
- Ser Criativo, não se limitar a seguir a receita, mas tentar melhorar a mesma, ou ainda melhor procurar criar as suas próprias receitas
- Não ter medo de errar, porque o sucesso só se alcança através de tentativas (um plano pode ajudar). Falhar é não tentar, é através dos erros que mais aprendemos.
- Ser Assertivo, só se aprende a ser, com a experiencia adquirida, os erros conta muito para a aquisição dessa experiencia.
- Não esperar que os outros tomem a iniciativa, todos podemos ser líderes, nem que seja de nós próprios, o que já é um ótimo começo.
- Saber delegar, em vez dar ordens devemos procurar pedir opiniões (envolver os outros).
- Reconhecer o mérito próprio (mais fácil), e o dos outros (mais complicado).
- Perceber que independentemente do papel desempenhado por alguém dentro de uma organização, o que cada um faz tem valor, tantos os “Chefes” como os “índios”, são essenciais para o sucesso.
- Saber ouvir, sem preconceitos.
- A confrontação saudável de diferença de opiniões enriquece os envolvidos.
- Fomentar a amigabilidade, a qualidade ou caráter de quem é amigável, mesmo para quem lhe é estranho, quem não conhece.
- Conseguir potenciar o espírito de equipa/comunidade, através da vivência conjunta de experiencias.
- Ser hábil ou talentoso, conseguir por em prática as técnicas e conhecimentos adquiridos.
- Dar atenção ao individuo dentro do grupo, o pormenor conta tanto ou mais do que o conjunto.

- Melhorar os canais de comunicação existentes, conseguir eliminar o ruído de fundo, saber usar o vocabulário indicado, melhorar as técnicas de comunicação.

Poderíamos continuar a adicionar muito mais "coisas" a esta lista, mas o objetivo principal é de conseguir criar formas metódicas, e se possível de alguma forma mensuráveis, de forma a conseguir acompanhar o progresso.

Espero de futuro poder concretizar este meu plano. Sei que provavelmente para o conseguir realizar como deve de ser, terei de aprofundar os meus conhecimentos em varias áreas, para além de achar que me deva rodear (associar) de um grupos de pessoas que sejam tão ou mais apaixonadas e ambiciosas como eu, de forma a conseguir criar a fundamentação científica necessária, para depois conseguir no futuro aplicar a *Framework,* na prática. Para isso, estas pessoas devem de ser de preferência de diferentes áreas do conhecimento, como por exemplo das áreas:

- Da psicologia e sociologia
- Da gestão de projetos e organizações
- Do desenvolvimento de *Software*
- Ou de outras áreas (todas as contribuições são bem vindas!)

Procurando desta forma solidificar "tecnicamente" as abordagens a serem tomadas. As quais, eu acredito seriamente que, se bem formuladas e postas em pratica, consigam ajudar todos os envolvidos a "**progredir verdadeiramente**"!

Bibliografia

Andriole, S.J., 2011. *It's All about the People: Technology Management that Overcomes Disaffected People, Stupid Processes, and Deranged Corporate Cultures*, CRC Press.

Anon, 2014. History of communication. *Wikipedia, the free encyclopedia*. Available at: http://en.wikipedia.org/w/index.php?title=History_of_communication&oldid=595782510.

Atkinson, R., 1999. Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International journal of project management*, 17(6), pp.337–342.

Baetjer, H., 1997. *Software as capital: An economic perspective on software engineering*, IEEE Computer Society Press.

Basili, V.R. & Larman, C., 2003. Iterative and incremental development: A brief history. *IEEE Computer Society*.

Basili, V.R. & Turner, A.J., 1975. Iterative enhancement: A practical technique for software development. *Software Engineering, IEEE Transactions on*, (4), pp.390–396.

bellis, M., Johannes Gutenberg - Printing Press. *About.com Inventors*. Available at: http://inventors.about.com/od/gstartinventors/a/Gutenberg.htm.

Benington, H.D., 1987. Production of large computer programs. In *ICSE*. pp. 299–310.

Bernstein, L., 1996. Foreword: Importance of software prototyping. *Journal of Systems Integration*, 6(1), pp.9–14.

Betts, M., 2003. Why IT projects fail.

Boehm, B., 2000. Requirements that handle IKIWISI, COTS, and rapid change. *Computer*, 33(7), pp.99–102.

Boehm, B.W., 1988. A spiral model of software development and enhancement. *Computer*, 21(5), pp.61–72.

Boehm, B.W., 1987. Software process management: lessons learned from history. In *Proceedings of the 9th international conference on Software Engineering*. IEEE Computer Society Press, pp. 296–298.

Brooks, F., 1995. *The Mythical Man-Month, Anniversary Edition: Essays on Software Engineering*, Pearson Education.

Brooks Jr, F.P., 1995. *The Mythical Man-Month, Anniversary Edition: Essays on Software Engineering*, Pearson Education.

Canario, R., 2006. *A Escola tem Futuro?*, Grupo A.

Cockburn, A., 1999. Software development as a cooperative game. In *Talk at 1999 ObjectActive conference, MidRange, South Africa*.

Cockburn, A., 2004. The end of software engineering and the start of economic-cooperative gaming. *Computer Science and Information Systems*, 1(1), pp.1–32.

Cockburn, A. & Highsmith, J., 2001. Agile software development, the people factor. *Computer*, 34(11), pp.131–133.

Crespi, V., Galstyan, A. & Lerman, K., 2008. Top-down vs bottom-up methodologies in multi-agent system design. *Autonomous Robots*, 24(3), pp.303–313.

Date, C.J., 2000. *What not how: the business rules approach to application development*, Addison-Wesley Professional.

DeMarco, T. & Lister, T.R., 1987. *Peopleware*, Dorset House Pub.

Eckstein, J. & Baumeister, H., 2004. *Extreme Programming and Agile Processes in Software Engineering: 5th International Conference, XP 2004, Garmisch-Partenkirchen, Germany, June 6-10, 2004, Proceedings*, Springer.

Evans, D.L., 1999. A Critical Examination of Claims Concerning The Impact of Print. Available at: http://www.aber.ac.uk/media/Students/dle9701.html.

Fowler, M. & Highsmith, J., 2001. The agile manifesto. *Software Development*, 9(8), pp.28–35.

Gottesdiener, E., 1995. Rad realities: beyond the hype to how rad really works. *Application Development Trends*, 2(8).

Guisepi, R.A., 1999. Writing. *The International History Project*. Available at: http://history-world.org/writing.htm.

Hall, D., 1987. Fail Fast, Fail Cheap. *Blomberg businessweek (Magazine)*. Available at: http://www.businessweek.com/stories/2007-06-24/fail-fast-fail-cheap.

Kaviratna, H., 1971. Unbroken chain of oral tradition. *Retrieved November*, 17, p.2009.

Lafley, A.G., 2011. I Think of My Failures as a Gift. *Harvard Business Review*.

Mark, J.J., 2011. Cuneiform Writing. *Ancient History Encyclopedia*. Available at: http://www.ancient.eu.com/image/93/.

Martin, J., 1991. *Rapid application development*, Macmillan Publishing Co., Inc.

Medicare, C. for, Services, M. & others, 2012. Selecting a development approach.

Al Neimat, T., 2005. Why IT projects fail. *online in Project Perfect Project Management Software, available at http://www. projectperfect. com. au/info_it_projects_fail. php (retrieved 8/2011)*.

Pressman, R., 2010. *Software Engineering: A Practitioner's Approach*, McGraw-Hill, Inc.

Pressman, R.S., 2009. *Software Engineering: A Practitioner's Approach*, McGraw-Hill.

Priberam, 2014. Significado / definiçãoo de maquiavélico no Dicionário Priberam da Língua Portuguesa. Available at: http://www.priberam.pt/dlpo/maquiav%C3%A9lico.

Richards, R., The Disadvantages of Written Communication. *eHow*. Available at: http://www.ehow.com/info_8130487_disadvantages-written-communication.html.

Royce, W.W., 1970. Managing the development of large software systems. In *proceedings of IEEE WESCON*. Los Angeles.

Sommerville, I., 2010. Software Engineering, 9/E.

Sommerville, I., 1996. Software process models. *ACM Computing Surveys (CSUR)*, 28(1), pp.269–271.

Standish Group, I., 2013. The CHAOS report.

Wiegers, K.E., 1996. *Creating a software engineering culture*, Pearson Education.

Wiegers, K.E., 1995. In search of excellent requirements. *The Journal of the Quality Assurance Institute*, 1(995), p.1.

Wiegers, K.E. & Beatty, J., 2013. *Software requirements, Third Edition.*, Microsoft Press.

Zwiers, J. et al., 1995. Synthesizing different development paradigms: Combining top-down with bottom-up reasoning about distributed systems. In *Foundations of Software Technology and Theoretical Computer Science*. Springer, pp. 80–95.