ACOMPANHAMENTO DA PRODUTIVIDADE DE UM TIME ÁGIL ATRAVÉS DO COCOMO II

Trabalho de Conclusão de Curso

Engenharia da Computação

Jair Medeiros Ferreira Filho
Orientador: Prof. Joabe Bezerra de Jesus Júnior

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

JAIR MEDEIROS FERREIRA FILHO

ACOMPANHAMENTO DA PRODUTIVIDADE DE UM TIME ÁGIL ATRAVÉS DO COCOMO II

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, maio de 2022.

Filho, Jair Medeiros Ferreira
Acompanhamento da produtividade de um time ágil através do COCOMO II / Jair Medeiros Ferreira Filho. - Recife, 2022.
xiv, 45 f. :il. ; 29 cm.

Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação) - Universidade de Pernambuco - UPE, Escola Politécnica de Pernambuco, Recife, 2022

Orientador (a): Profº Mscº. Joabe Bezerra de Jesus Júnior.

1. Estimativas de produtividade. 2. Metodologia ágil. 3. Mineração de Dados. I. Acompanhamento da produtividade de um time ágil através do COCOMO II. II. Bezerra, Joabe. III. Universidade de Pernambuco.

MONOGRAFIA DE FINAL DE CURSO

Avaliação Final (para o presidente da banca)*

No dia 27/5/2022, às 14h45min, reuniu-se para deliberar sobre a defesa da monografia de conclusão de curso do(a) discente **JAIR MEDEIROS FERREIRA FILHO**, orientado(a) pelo(a) professor(a) **JOABE BEZERRA DE JESUS JÚNIOR**, sob título ACOMPANHAMENTO DA PRODUTIVIDADE DE UM TIME ÁGIL ATRAVÉS DO COCOMO II, a banca composta pelos professores:

ALEXANDRE MAGNO ANDRADE MACIEL (PRESIDENTE)
JOABE BEZERRA DE JESUS JÚNIOR (ORIENTADOR)

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

x Aprovada ☐ Aprovada com Restrições* ☐ Reprovada

e foi-lhe atribuída nota: 9,0 (nove)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O(A) discente terá oito dias para entrega da versão final da monografia a contar da data deste documento.

gov.br Documento assinado digitalmente
ALEXANDRE MAGNO ANDRADE MACIEL
Data: 27/05/2022 15:53:57-0300
Verifique em https://verificador.iti.br

AVALIADOR 1: Prof (a) **ALEXANDRE MAGNO ANDRADE MACIEL**

Joabe Bezerra de Jesus Júnior

AVALIADOR 2: Prof (a) **JOABE BEZERRA DE JESUS JÚNIOR**

AVALIADOR 3: Prof (a)

* Este documento deverá ser encadernado juntamente com a monografia em versão final.

Dedico este trabalho para todos que me auxiliaram de alguma forma durante a graduação, especialmente meus familiares, amigos e professores.

AGRADECIMENTOS

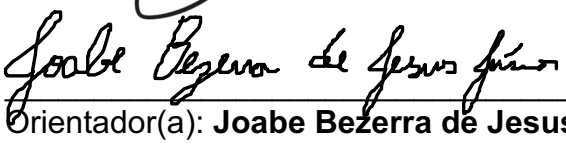
A todos aqueles que contribuíram, de alguma forma, para a realização deste trabalho. Agradeço a todos que me ajudaram durante toda a graduação, principalmente minha mãe, minha avó e minha tia. Sendo elas a base do profissional que me tornei, através de todo o investimento e dedicação. Além disso, agradeço a minha noiva que foi essencial para me motivar na reta final e até mesmo na elaboração deste trabalho. Agradeço também aos colegas e professores que tive contato durante toda a caminhada, por me auxiliar e dar apoio, de alguma forma, para que eu pudesse chegar até aqui.

Autorização de publicação de PFC

Eu, **Jair Medeiros Ferreira Filho** autor(a) do projeto de final de curso intitulado: **ACOMPANHAMENTO DA PRODUTIVIDADE DE UM TIME ÁGIL ATRAVÉS DO COCOMO II;** autorizo a publicação de seu conteúdo na internet nos portais da Escola Politécnica de Pernambuco e Universidade de Pernambuco.

O conteúdo do projeto de final de curso é de responsabilidade do autor.

Jair Medeiros Ferreira Filho



Orientador(a): **Joabe Bezerra de Jesus Júnior**

Coorientador(a):

Prof, de TCC: **Daniel Augusto Ribeiro Chaves**

Data: 27/5/2022

RESUMO

Dispostas a trocar qualidade e o compromisso com requisitos, atualmente, várias empresas adotaram o desenvolvimento ágil para trazer um cadenciamento mais eficiente do produto que precisam entregar. Para que haja uma melhor compreensão dos fatores e intermediários que afetam a produtividade do time ágil, todo o processo pode ser analisado como um processo comportamental através de métricas de software. Este trabalho tem como objetivo analisar a utilização do modelo COCOMO II ou *Constructive Cost Model II* como uma medida de produtividade para compreensão da eficiência de produção de um time ágil. O mesmo foi desenvolvido em uma empresa onde utiliza o método Kanban para acompanhamento do desenvolvimento. A metodologia adotada inspirou-se no CRISP-DM, sendo este um processo de mineração de dados, através de fases de obtenção, preparação e compreensão dos dados coletados de ferramentas auxiliares, *GitHub* e Jira. Ao término destas fases de análise dos dados, obteve-se uma base de dados que pudesse ser utilizada para comparação com os resultados gerados pelo modelo, utilizando pessoa-mês como medida. Dessa forma, o presente trabalho obteve resultados comparativos pouco satisfatórios e fora do contexto, devido ao escopo mais limitado, de um único time, com um único projeto. Entretanto, vale ressaltar que este trabalho procura estruturar um processo automatizado para aplicação do modelo dentro de um time ágil, permitindo uma refinamento quanto ao modelo e sua calibração para uma realidade atual em relação a sua criação.

Palavras-chaves: Metodologia ágil, COCOMO II, CRISP–DM, Produtividade.

ABSTRACT

Willing to trade quality and commitment to requirements, several companies have now adopted agile development to bring more efficient cadence to the product they need to deliver. In order to have a better understanding of the factors and intermediaries that affect the productivity of the agile team, the whole process can be analyzed as a behavioral process through software metrics. This work aims to analyze the use of the COCOMO II or Constructive Cost Model II as a productivity measure to understand the production efficiency of an agile team. It was developed in a company that uses the Kanban method to monitor development. The adopted methodology was inspired by CRISP-DM, which is a data mining process, through phases of obtaining, preparing and understanding the data collected from auxiliary tools, GitHub and Jira. At the end of these data analysis phases, a database was obtained that could be used for comparison with the results generated by the model, using person-months as the measure. Thus, the present work obtained comparative results that were not very satisfactory and out of context, due to the more limited scope of a single team, with a single project. However, it is worth noting that this work seeks to structure an automated process for applying the model within an agile team, allowing a refinement of the model and its calibration to a current reality in relation to its creation

Keywords: Agile Methodology, COCOMO II, CRISP–DM, Productivity.

LISTA DE FIGURAS

FIGURA 1 – Projetos de software finalizados……..17

FIGURA 2 – Custos de alterações como uma função do tempo em desenvolvimento……18

QUADRO 1 – Os princípios dos métodos ágeis……...19

FIGURA 3 – Visão geral do sistema Kanban……20

FIGURA 4 – Método Kanban no contexto do fluxo de ponta a ponta……...21

FIGURA 5 – Projeção da incerteza de estimativas……..22

FIGURA 6 – Sub modelos do COCOMO…….……..24

QUADRO 2 – Fatores de escala…….28

QUADRO 3 – Multiplicadores de esforço para os modelos preliminar e pós-arquitetura……28

QUADRO 4 – Multiplicadores de esforço do modelo de projeto preliminar…….29

QUADRO 5 – Multiplicadores de esforço do modelo de projeto pós-arquitetura……30

FIGURA 7 – Decomposição da Metodologia CRISP–DM em quatro níveis para mineração de dados……..33

FIGURA 8 – Fases do atual modelo do processo CRISP-DM para mineração de dados……..34

FIGURA 9 – Fluxo para obtenção dos dados……...37

FIGURA 10 – Utilização do COCOMO nos processos RUP e modelo em cascata……41

FIGURA 11 – Exemplo de um quadro Kanban na fase de *Downstream*……..42

FIGURA 12 – Gráfico de KSLOCs por dias passados……43

FIGURA 13 – Clusters auto gerados a partir dos dados……44

QUADRO 6 – Fatores de escala utilizados para mensurações…….48

QUADRO 7 – Multiplicadores de esforço utilizados para mensurações……..48

QUADRO 8 – Dicionário parcial dos dados……..……...50

QUADRO 9 – Dicionário final dos dados……...52

FIGURA 14 – Gráfico de pessoa-mês por KSLOCs extraídos de dados reais……...53

FIGURA 15 – Gráfico de pessoa-mês por KSLOCs extraídos de dados estimados 54

LISTA DE SÍMBOLOS E/OU SIGLAS

CRISP–DM	*Cross Industry Standard Process for Data Mining*
COCOMO II	*Constructive Cost Model II*
SLOC	*Source Lines of Code*
KSLOC	*Kilo Source Lines of Code*
JIT	*Just-In-Time*
TPS	*Toyota Production System*

SUMÁRIO

1 INTRODUÇÃO……13

1.1 CONTEXTUALIZAÇÃO…… 13

1.2 PROBLEMA DE PESQUISA……14

1.3 OBJETIVOS…… 15

2 FUNDAMENTAÇÃO TEÓRICA……...16

2.1 METODOLOGIA ÁGIL……..16

2.1.1 Método Kanban…….. 19

2.2 TÉCNICAS DE ESTIMATIVA……...22

2.2.1 COCOMO II……23

2.2.1.1 Medição de Tamanho do Software por Linhas de Código……...24

2.2.1.2 Modelo de Composição de Aplicações……...25

2.2.1.3 Modelos de Projeto Preliminar e de Pós-arquitetura……26

2.2.1.4 Modelo de Reúso……31

2.2.2 CRISP–DM……..32

2.2.2.1 Entendimento do Negócio……33

2.2.2.2 Entendimento dos Dados……..34

2.2.2.3 Preparação dos Dados……..34

2.2.2.4 Modelagem……..35

2.2.2.5 Avaliação…… 35

2.2.2.6 Aplicação…… 35

3 MATERIAIS E MÉTODOS…… 36

3.1 MATERIAIS…… 36

3.1.1 Objetivo de Pesquisa……36

3.1.2 Ferramentas Utilizadas……..36

3.2 MÉTODOS..... 36

4 ACOMPANHAMENTO DA PRODUTIVIDADE DE UM TIME ÁGIL ATRAVÉS DO COCOMO II..... 38

4.1 ENTENDIMENTO DO NEGÓCIO: COCOMO II COMO MEDIDA DE PRODUTIVIDADE..... 39

4.2 ENTENDIMENTO DOS DADOS: APLICAÇÃO DO COCOMO II NO MÉTODO KANBAN..... 40

4.2.1 Compreensão dos dados..... 42

4.2.1.1 Análise Descritiva Dos Dados..... 42

4.2.1.2 Paradoxos Associados Com A Contagem Das Linhas De Código 44

4.3 PREPARAÇÃO DOS DADOS: FATORES DE ESCALA E MULTIPLICADORES DE ESFORÇO..... 47

4.4 MODELAGEM: MÉTODO DE MEDIÇÃO ORIENTADA PELOS DADOS..... 49

4.4.1 Obtenção dos Dados Iniciais..... 49

4.4.2 Preparação do Atributo Comparativo..... 51

4.5 AVALIAÇÃO: APLICAÇÃO DO COCOMO II..... 52

4.6 APLICAÇÃO: RESULTADOS..... 53

5 CONCLUSÃO..... 56

5.1 TRABALHOS FUTUROS..... 56

REFERÊNCIAS..... 57

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Com uma sociedade cada vez mais informatizada e dinâmica, a Engenharia de *software* enfrenta desafios contínuos para atender a crescente demanda por *softwares* mais complexos. Do mesmo modo, características como abstração e intangibilidade (SOMMERVILLE, 2016) são associadas à natureza do *software*, apesar do mesmo apresentar-se como uma “entidade descritiva, complexamente hierarquizada, cognitivo-linguística e histórica.” (FERNANDES, 2003, v. 55, p. 29–33).

Como tentativa para estruturar e ordenar o desenvolvimento, modelos de processo prescritivos são aplicados há anos (PRESSMAN, 2015). Entretanto, os mesmos são dependentes de um conjunto de processos predeterminados e uma documentação elaborada continuamente (NIKIFOROVA *et al*., 2009, p. 229–239). Consequentemente, o sucesso do projeto acaba prejudicado em contraste com a necessidade de um desenvolvimento mais acelerado e adepto às mudanças. Além disso, normalmente, o cliente não tem o conhecimento pleno do que deseja ser implementado no *software* ou sistema (LEAU *et al*., 2012, v. 37, p. 162–167) e modelos como os citados, geralmente, precisam da etapa de planejamento finalizada para iniciar o desenvolvimento do produto. Dessa forma, dispostas a trocar qualidade e o compromisso com requisitos (SOMMERVILLE, 2016), atualmente, várias empresas adotaram o desenvolvimento ágil para trazer um cadenciamento mais eficiente do produto que precisam entregar.

Com foco na adaptabilidade, o processo ágil ocorre de forma incremental (PRESSMAN, 2015), onde as fases dentro de um ciclo de vida de desenvolvimento são revisitadas continuamente. E, por requerer menos planejamento, com uma divisão em pequenas tarefas (SHARMA *et al*., 2012, v. 4, n. 5, p. 892), o *software* é entregue rapidamente aos clientes. Onde, logo após, os mesmos podem imediatamente propor mudanças e adaptações a serem implementadas futuramente.

1.2 PROBLEMA DE PESQUISA

Entretanto, na prática, os princípios básicos dos métodos ágeis às vezes são difíceis de alcançar (SOMMERVILLE, 2016). Como mencionado anteriormente, o desenvolvimento ágil se concentra na comunicação e participação do cliente, o que significa que as habilidades interpessoais e sociais são essenciais para o desenvolvimento de qualquer time (LEAU *et al*., 2012, v. 37, p. 162–167). Porém, muitos dos desenvolvedores são introvertidos e estão mais confortáveis trabalhando por conta própria (CAPRETZ, 2003, v. 58, n. 2, p. 207–214), portanto, ao inseri-los em um time ágil, podem não interagir bem com outros membros. Além disso, há uma redução significativa da quantidade de documentação, de forma que o próprio código atua como um documento (VIJAYASARATHY; TURK, 2008, v. 19, n. 2, p. 1–8). Com menos informações disponíveis, fica muito mais difícil para novos membros compreender os métodos reais seguidos para desenvolver o *software* (SHARMA *et al*., 2012, v. 4, n. 5, p. 892).

Diante disso, percebe-se que os times de desenvolvimento precisam de uma combinação complexa de fatores (DINGSØYR; LINDSJØRN, 2013, v. 149, p. 46–60) para trabalhar com eficácia. Portanto, para um melhor entendimento de todas as consequências envolvidas ao projeto e, para um melhor auxílio quanto ao controle e melhoria do processo é necessário realizar medições (FENTON; BIEMAN, 2020).

Uma melhor compreensão dos fatores e intermediários que afetam a produtividade do time pode ajudar a determinar onde concentrar os esforços de gerenciamento para aumentar a produtividade (FATEMA; SAKIB, 2017, p. 737–742). Nesse sentido, o processo ágil pode ser analisado como um processo comportamental (DESTEFANIS *et al*., 2016, v. 2, p. e73) através de métricas de software, isto é, *software productivity*, para que haja uma maior sincronização entre o próprio time ágil para alcançar os objetivos esperados (LEFFINGWELL, 2007).

Como medida de produtividade pode-se utilizar modelos de estimativa ao contrário de utilizar dados históricos (DE RORE *et al*., 2008) para entendimento dos fatores que influenciam a produtividade.

1.3 OBJETIVOS

Desta forma, o presente trabalho tem como objetivo analisar a utilização do modelo COCOMO II (BOEHM, 2000) ou *Constructive Cost Model II* como uma ferramenta para compreensão da eficiência de produção de um time ágil. Através da metodologia CRISP-DM, sendo este um processo de mineração de dados (SHEARER, 2000, v. 5, n. 4, p. 13–22), foi realizada uma análise quanto aos dados coletados por uma empresa privada a partir de APIs. O modelo utiliza S*LOC* ou *Source Line of Codes* como medida de tamanho, onde foi possível realizar contagens automaticamente. E, apesar de apresentar paradoxos de produtividade (JONES, 1986), caso haja um ambiente onde exclui-se os supracitados paradoxos, o mesmo torna-se válido (SYMONS, 1988, v. 14, n. 1, p. 2–11).

2 FUNDAMENTAÇÃO TEÓRICA

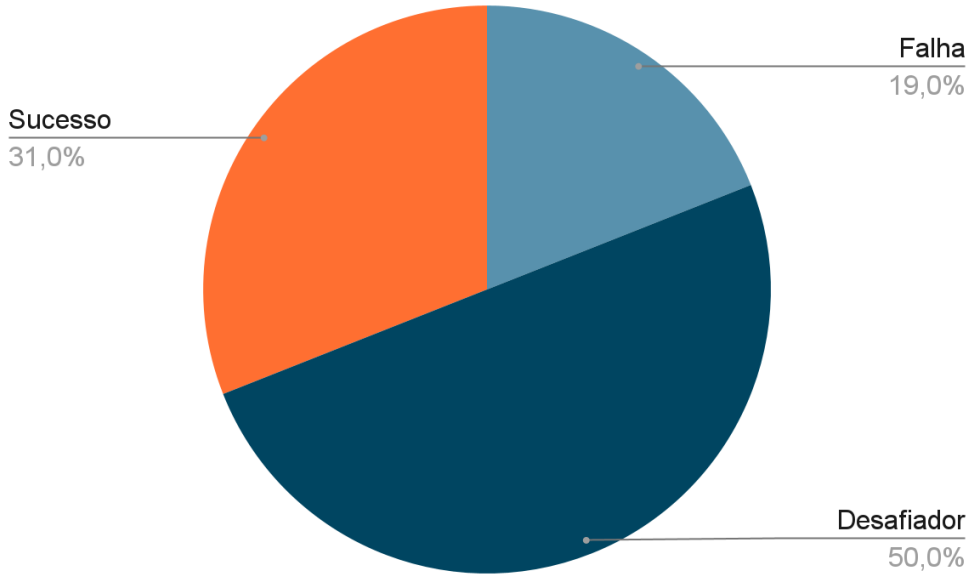
2.1 METODOLOGIA ÁGIL

Nos dias de hoje, *softwares* fazem parte de quase todos os contextos e operações presentes na sociedade. Para isso, os mesmos precisam ser desenvolvidos cada vez mais rápidos para obterem proveito de novas oportunidades e responder às pressões competitivas (SOMMERVILLE, 2016). Neste contexto moderno e dinâmico que estamos inseridos, é praticamente impossível prever como um *software* vai evoluir com o tempo. As condições de mercado e necessidades dos usuários mudam rapidamente (PRESSMAN, 2015), ou seja, é difícil obter requisitos completos de um *software* para ser considerado estável.

Processos de desenvolvimento, onde possuem etapas para especificar completamente os requisitos, e, em seguida, construir o *software* tendem a não estarem adaptados a velocidade necessária para o desenvolvimento de hoje. Desta forma, processos convencionais, herdados de outras áreas da Engenharia, como de *em cascata*, acabam entregando softwares atrasados (SOMMERVILLE, 2016).

Como provável consequência, atualmente, conforme exposto pela Figura 1, somente cerca de 30% dos projetos são finalizados com sucesso (JOHNSON, 2020), mesmo com uma crescente adoção das metodologias ágeis, o que pode indicar que processos convencionais ainda estão sendo utilizados.

FIGURA 1 – Projetos de software finalizados



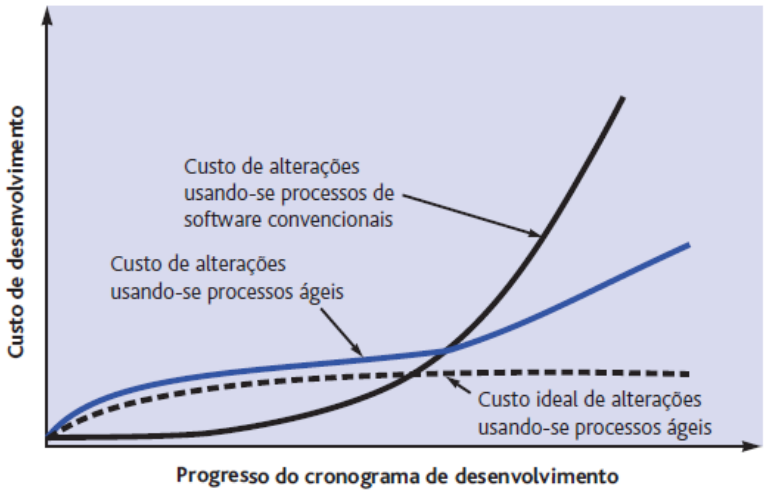
Fonte: JOHNSON, 2020

Com cada vez mais empresas dispostas a trocar a qualidade e o compromisso com requisitos por uma implantação mais rápida (SOMMERVILLE, 2016), a metodologia ágil tornou-se uma alternativa viável quando comparada com a Engenharia de *Software* estabelecida anteriormente. E, concebida para rapidamente produzir *softwares* de valor, a agilidade, no âmbito de Engenharia de *software*, enfatiza a entrega rápida e diminui a importância das documentações, com objetivo reduzir a burocracia do processo (SOMMERVILLE, 2016). Além disso, incentiva a estruturação e as atitudes em time que podem tornar a comunicação mais fácil, tanto entre membros da equipe, quanto com o cliente. Com isso, para que o mesmo possa propor alterações e novos requisitos é necessário uma abordagem incremental para todos os processos associados ao *software* (SOMMERVILLE, 2016).

Em linhas gerais, esta construção abordada acima caracteriza-se como uma time ágil, onde o mesmo reconhece que o *software* é desenvolvido por indivíduos trabalhando em equipe e que suas habilidades e capacidade de colaborar estão no cerne do sucesso do projeto (PRESSMAN, 2015). Desta forma, permite que uma equipe, imersa em um processo ágil, assimile as alterações realizadas posteriormente em um projeto, sem um impacto significativo (BECK; ANDRES, 2005). Através deste contexto, é possível haver uma estabilização quanto ao custo

da curva de mudança (Figura 2, curva em linha azul), em comparação com processos convencionais.

FIGURA 2 – Custos de alterações como uma função do tempo em desenvolvimento



Fonte: PRESSMAN, 2015

A filosofia por trás da metodologia ágil é estruturada diretamente do manifesto ágil, onde afirma-se que:

> Estamos descobrindo melhores maneiras de desenvolver softwares, fazendo-o e ajudando outros a fazê-lo. Através desse trabalho, valorizamos mais:
>
> - Indivíduos e interações do que processos e ferramentas.
> - Software em funcionamento do que documentação abrangente.
> - Colaboração do cliente do que negociação de contrato.
> - Respostas a mudanças do que seguir um plano.
>
> Ou seja, embora itens à direita sejam importantes, valorizamos mais os que estão à esquerda. (BECK *et al*., 2001).

Através das premissas apresentadas várias metodologias ágeis foram desenvolvidas e aperfeiçoadas durante o tempo, de forma que, levou a uma certa integração com métodos mais tradicionais (SOMMERVILLE, 2016), o que permitiu uma maior adesão entre as empresas. E, mesmo que cada abordagem proponha diferentes processos para alcançar o conceito de desenvolvimento e entrega incremental, os mesmos compartilham um conjunto de princípios (Quadro 1), com base no manifesto (SOMMERVILLE, 2016).

Quadro 1 – Os princípios dos métodos ágeis

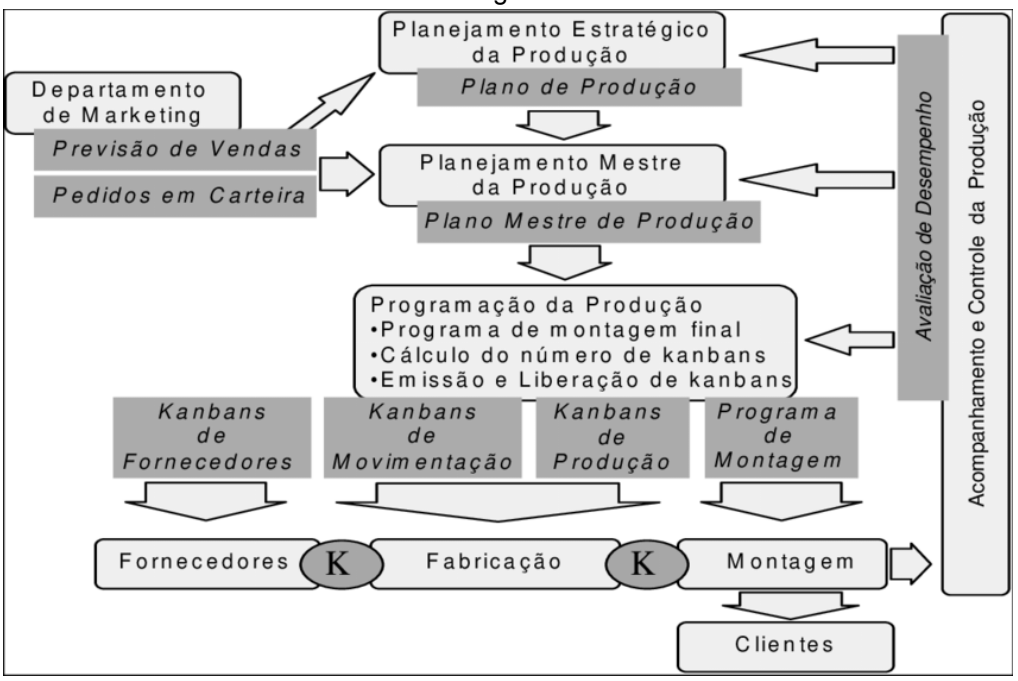
Princípios	Descrição
Envolvimento do cliente	Os clientes devem estar intimamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar suas iterações.
Entrega incremental	O software é desenvolvido em incrementos com o cliente, especificando os requisitos para serem incluídos em cada um.
Pessoas, não processos	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Os membros da equipe devem desenvolver suas próprias maneiras de trabalhar, sem processos prescritivos.
Aceitar as mudanças	Deve-se ter em mente que os requisitos do sistema vão mudar. Por isso, projete o sistema de maneira a acomodar essas mudanças.
Manter a simplicidade	Focalize a simplicidade, tanto do software a ser desenvolvido quanto do processo de desenvolvimento. Sempre que possível, trabalhe ativamente para eliminar a complexidade do sistema.

Fonte: SOMMERVILLE, 2016

2.1.1 Método Kanban

Com o propósito de ser menos prescritivo (KNIBERG; SKARIN, 2010), o método Kanban tornou-se uma alternativa viável para utilização das metodologias ágeis dentro de um time. Em contrapartida a métodos mais populares como Scrum e XP, o Kanban tem como característica evidenciar os problemas existentes no processo (SILVA *et al*., 2012, p. 715–725).

FIGURA 3 – Visão geral do sistema Kanban



Fonte: JULIA; FORNO, 2022

Concebido como parte do sistema de produção *Just-In-Time* (JIT) da *Toyota Production System* (TPS) para controlar a fabricação de automóveis, originalmente o método Kanban determinava a demanda que iria ditar o ritmo da produção (SILVA *et al*., 2012, p. 715–725). Além disso, era também chamado de “Método de Supermercado”, e dentro da Toyota, o mesmo foi dividido em duas fases, manufatura e distribuição (Figura 3), sendo utilizados para gerenciar partes no processo de produção (KIROVSKA; KOCESKI, 2015, v. 3, n. 3, p. 25–34).

Kanban é uma palavra de origem japonesa que significa, aproximadamente, cartão de sinalização, onde, no contexto da Toyota, era utilizado para sinalizar quando iniciar a manufatura. Esta característica de ser um método muito mais visual e sinalizador, acabou estimulando ainda mais as equipes a adotarem (SILVA *et al*., 2012, p. 715–725). Entretanto, para o contexto do desenvolvimento de *software*, o mesmo não é uma produção ou uma atividade de fabricação (REEVES, 1992, v. 2, n. 2, p. 14–12) então princípios precisam ser determinados para um contexto mais dinâmico.

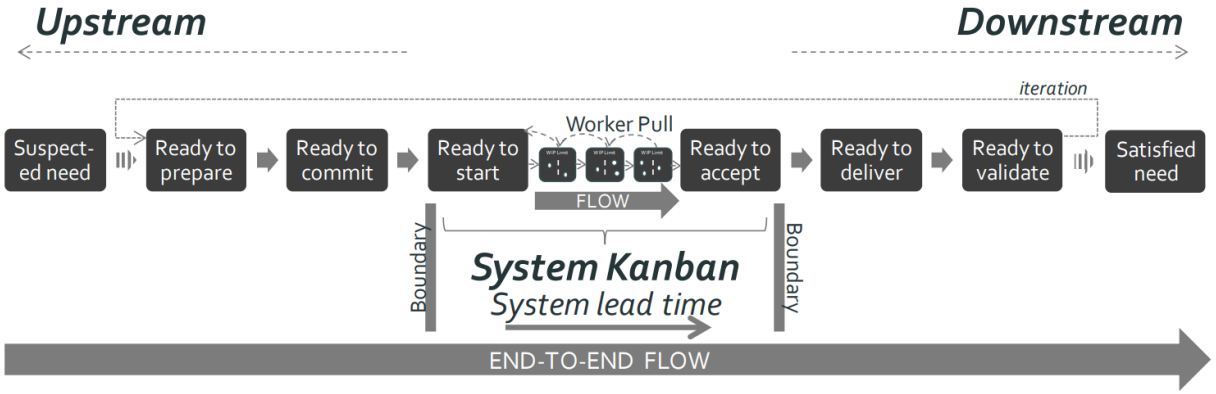
O início do envolvimento do Kanban no processo de desenvolvimento de *software* foi através de David J. Anderson (ANDERSON, 2010), onde, princípios básicos foram estabelecidos para um processo de adoção bem sucedido do método:

- Começar com o que tem agora – sendo este o progresso atual.
- Manter uma abordagem progressiva para mudar e melhorar.
- Respeitar as funções e responsabilidades atuais da equipe ou organização.

Sendo um método pouco prescritivo, o Kanban acaba tornando-se muito adaptativo e empírico (SILVA *et al*., 2012, p. 715–725), de forma que, melhorias e adaptações precisam existir e serem testadas para que o processo possa se concretizar. Dessa forma, de acordo com os princípios citados acima e, através de observações em empresas que adotaram o Kanban com sucesso, outros três princípios essenciais foram determinados (KNIBERG; SKARIN, 2010):

1. Visualize o fluxo de trabalho atual.
2. Limite a carga de trabalho.
3. Acompanhe e gerencie o trabalho envolvido.

FIGURA 4 – Método Kanban no contexto do fluxo de ponta a ponta



Fonte: STEYAERT, 2018

Com um fluxo de trabalho identificado, para o contexto do time onde o método Kanban está sendo inserido, para limitar o fluxo de trabalho é necessário explicitar quantos itens de trabalho devem estar em cada uma das etapas do processo (SILVA *et al*., 2012, p. 715–725). Todos têm apenas uma tarefa de cada vez e o trabalho é "puxado" depois de terminado, ou seja, apenas quando um item sair de uma etapa é que esta etapa poderá receber outro item (KNIBERG; SKARIN, 2010). Em alusão com as fases abordadas anteriormente pela Toyota, manufatura e distribuição, para desenvolvimento de *software*, temos a *Upstream* e o *Downstream* (Figura 4). Sendo a primeira, a fase de idealização e estruturação das demandas, e a segunda,

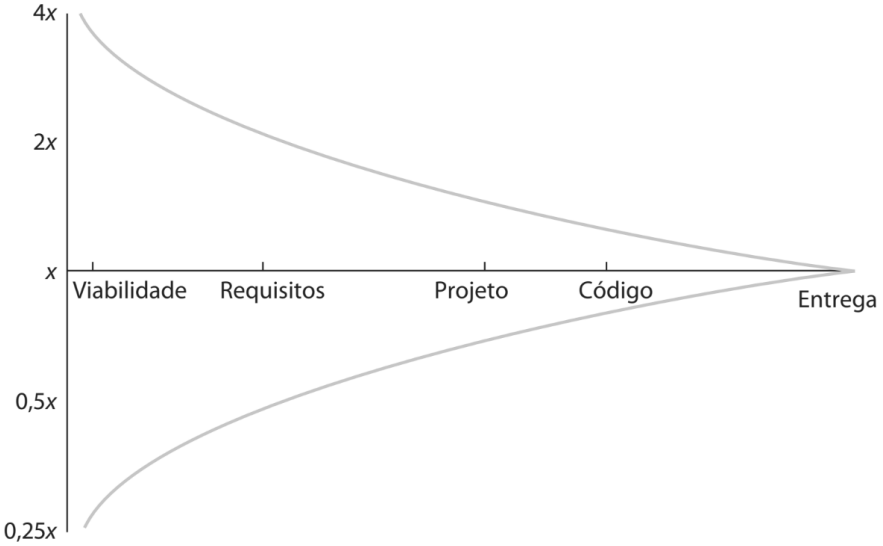
associada às etapas que terão limitação de fluxo de trabalho, sendo inteiramente ligadas ao desenvolvimento.

Dessa forma, temos o método Kanban com um processo contínuo baseado no fluxo de trabalho, onde controla as entradas de itens de trabalho e a vazão que é dada de acordo com o limite estabelecido (SILVA *et al*., 2012, p. 715–725), através do *Downstream*.

2.2 TÉCNICAS DE ESTIMATIVA

Com as diversas variáveis que podem alterar o processo de desenvolvimento de *software* e para manter o cliente sempre atualizado quanto às expectativas envolvidas com produto final, é importante realizar estimativas. Entretanto, estimar projetos de *software* é difícil (SOMMERVILLE, 2016), e quanto mais inicial for o estágio do projeto, associado com o possível desconhecimento das tecnologias, mais improvável torna-se estimar com precisão os custos de desenvolvimento.

FIGURA 5 – Projeção da incerteza de estimativas



Fonte: SOMMERVILLE, 2016

A estimativa possui um importante papel para para definir o orçamento do projeto, entretanto, é necessário um julgamento para determinar o esforço, ou as características de projeto e de produto (SOMMERVILLE, 2016). E, sendo realizado no início, as estimativas podem ter uma margem de erro de 0,25x até 4x do esforço

real medido (BOEHM *et al.*, 1995). Porém, à medida que o projeto avança, os resultados vão se tornando mais precisos (Figura 5).

Apesar das incertezas apresentadas, as empresas ainda precisam fazer estimativas possibilitando, ajustar projetos para se adequar a orçamentos gerados a partir dos resultados. Dessa forma, há dois tipos de técnicas que podem ser utilizadas para gerar resultados:

- Baseadas em experiências: Os requisitos para os esforços baseiam-se na experiência da equipe em projetos similares. Entretanto, para projetos novos e fora do domínio consolidado, poderá ser mais difícil produzir os custos precisos e estimativas de cronograma (SOMMERVILLE, 2016).
- Modelos empíricos de estimativa: Através de fórmulas derivadas empiricamente, esta é uma alternativa onde é calculado o esforço do projeto com base em atributos como tamanho e características do processo. Além disso, o modelo é testado aplicando dados coletados de projetos completos e comparando resultados reais com os previstos (PRESSMAN, 2015).

2.2.1 COCOMO II

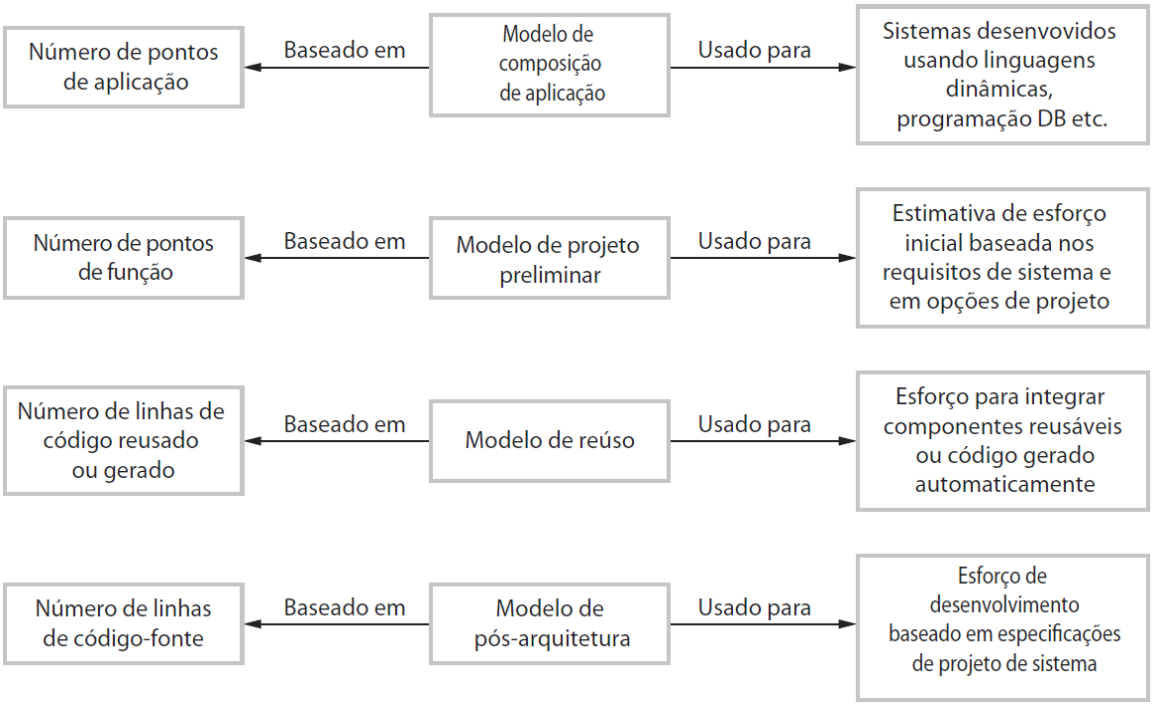
Conforme abordado acima, estimar sempre vai ser um desafio, principalmente quando associado ao esforço, o cronograma e os custos de um projeto de *software*. Para solucionar, ou amenizar este contexto, vários modelos empíricos foram criados, e o COCOMO II, ou *Constructive Cost Model* (modelo construtivo de custo) II tornou-se destaque a partir da sua versão original, sendo um dos modelos de estimativa de custo de software mais amplamente usado e discutido no setor (PRESSMAN, 2015).

O modelo original, também chamado de COCOMO81, através de fórmulas obtidas por meio da coleta de dados de 63 projetos de *software* finalizados, vinculou o tamanho do produto, os projetos e a equipe ao esforço para desenvolver o *software* (SOMMERVILLE, 2016). Dessa forma, era possível estimar o número de pessoas-mês necessários para desenvolver um produto através do esforço envolvido. Esta nova versão, além de absorver as premissas originais abordadas acima, levou em consideração métodos mais contextualizados, com a época em que

foi desenvolvido, para o desenvolvimento de *software*. Para um projeto desenvolvido com o modelo de cascata, o esforço incluído na estimativa começa quando a revisão dos requisitos do software, SRR é concluída e termina quando a revisão de aceitação do software, SAR, é concluída (DE RORE, 2008). Quando um processo de desenvolvimento em espiral é usado, o esforço estimado começa com a revisão dos objetivos do ciclo de vida, LCO, e termina com a capacidade operacional inicial, IOC.

Além disso, assim como seu predecessor, o COCOMO II, na realidade, incorpora uma hierarquia de sub modelos (Figura 6), que aborda diferentes fases do processo de desenvolvimento do *software*, produzindo estimativas detalhadas.

FIGURA 6 – Sub modelos do COCOMO



Fonte: SOMMERVILLE, 2016

2.2.1.1 Medição de Tamanho do Software por Linhas de Código

Assim como todos os modelos de estimativa para *software*, os sub modelos do COCOMO II exigem informações de tamanho do sistema (PRESSMAN, 2015), entretanto, mensurar um produto pode ser bastante complexo. Projetos geralmente são compostos de código novo, reusados e traduzidos automaticamente. O COCOMO trata cada tipo de código de um modo diferente, mas todos precisam da

contagem de linhas de código, e existem várias formas para estimar (BOEHM, 2000).

De acordo com a fórmula de cálculo utilizada pelo COCOMO II, o tamanho do software é expresso em milhares de linhas de código-fonte (KSLOC) para medir o esforço associado ao desenvolvimento. Além disso, historicamente, as linhas de código (SLOC) têm sido a primeira medida de tamanho do software (DE RORE, 2008). Entretanto, esta forma de mensuração possui alguns problemas e paradoxos associados (JONES, 1986).

Não há uma definição universal para o que exatamente é uma linha de código. Desde linhas de execução de código, comentários ou *data definitions*, com essa diferença de contagem, é possível haver diferença de 5 para 1 de uma forma de contagem para a outra (JONES, 1986).

Há uma grande dependência da implementação do *software* quando a mensuração das linhas de código. Com diferentes tipos de linguagens de programação, foi desenvolvido uma tabela de conversão entre as linguagens (BOEHM, 2000). Além disso, o estilo, padrões utilizados e a complexidade para o desenvolvimento podem influenciar na contagem, de forma que um código não estruturado pode demonstrar que foi produtivo do que um código desenvolvido após algum pensamento bem estruturado (DE RORE, 2008).

Entretanto, este método de contagem de SLOCs ainda pode ser válido de ser utilizado. Se for possível criar um contexto onde os problemas acima sejam excluídos, então o método utilizado com o COCOMO pode ser considerado matematicamente correto (DE RORE, 2008). Além disso, o supracitado método é uma possível alternativa para automatização do processo de contagem e com o mínimo possível de esforço para coletar os dados necessários, devido, principalmente, às ferramentas existentes.

2.2.1.2 Modelo de Composição de Aplicações

Com o objetivo de alinhar as estimativas para projetos desenvolvidos pela composição de componentes existentes, este modelo é baseado na aplicação ponderada de pontos (SOMMERVILLE, 2016). A estimativa é ajustada de acordo

com a dificuldade de desenvolvimento, de forma que a produtividade depende da experiência e da capacidade do desenvolvedor (BOEHM, 2000).

Para uma estimativa mais assertiva, é importante considerar a composição da aplicação com um significativo reúso de software, e ajustar a mensuração, visto que alguns pontos de aplicação no *software* podem ser implementados usando componentes reusáveis. Dessa forma, a fórmula final para cálculo do esforço em protótipos de sistema será (SOMMERVILLE, 2016):

$$PM = (NAP * (1 - \%reúso/100))/PROD \quad (1)$$

Com isto, PM significa o resultado da estimativa de esforço em pessoa-mês. NAP é o número total de pontos de aplicação do sistema entregue. Além disso, $\%reúso$ é uma estimativa da quantidade de código *reusado* no desenvolvimento e $PROD$ é a produtividade de pontos de aplicação.

Assim como este modelo, todos a seguir resultam em pessoa-mês como representação da estimativa de esforço. Uma pessoa-mês é o tempo que uma pessoa passa trabalhando no projeto de desenvolvimento de *software* por um mês. COCOMO II trata o número de pessoas-horas por pessoa-mês (PH/PM) como um valor nominal de 152 horas por pessoa-mês. Este número exclui o tempo dedicado a feriados, férias e folgas nos fins de semana (BOEHM, 2000).

2.2.1.3 Modelos de Projeto Preliminar e de Pós-arquitetura

Utilizados no desenvolvimento para gerar aplicações, integrações entre sistemas ou desenvolvimento de Infra-estruturas (BOEHM, 2000), ambos os modelos são os mais abordados referente ao COCOMO II. Além disso, compartilham a mesma abordagem para a mensuração do produto (incluindo a reutilização) e para os fatores de escala.

O modelo de projeto preliminar pode ser usado durante os estágios iniciais de um projeto, antes do projeto de arquitetura detalhado para o sistema estar disponível (SOMMERVILLE, 2016). Utilizado para explorar alternativas de arquitetura ou estratégias incrementais de desenvolvimento dos requisitos de usuários, o modelo pode ser considerado de alto nível, devido ao nível geral de informação e precisão

disponível (BOEHM, 2000). Além disso, nesta fase do projeto, presume-se que os requisitos de usuários foram acordados, assim como os passos iniciais do projeto foram iniciados. Dessa forma, é necessário uma estimativa de custo rápida e aproximada (SOMMERVILLE, 2016), com hipóteses simplificadoras devido às informações de alto nível.

O modelo de pós-arquitetura pode ser utilizado associado com projeto de arquitetura preliminar, que deve ser finalizado anteriormente (SOMMERVILLE, 2016). Com informações mais completas quanto a decomposição do sistema, é possível fazer uma estimativa mais precisa do tamanho do projeto, de forma que este modelo pode ser considerado o mais detalhado.

Ambas as estimativas citadas são baseadas em uma fórmula, onde utiliza-se o KSLOC como medida de tamanho, e uma série de fatores de custo para estimar, em pessoa-mês, a quantidade de esforço necessária para desenvolver um projeto (BOEHM, 2000):

$$PM = A * Tamanho^{E} * \prod_{i=1}^{n} EM_i \quad (2)$$

onde,

$$E = B + 0.01 * \sum_{j=1}^{5} SF_j \quad (3)$$

Através do grande conjunto de dados obtidos pela calibração de 161 projetos finalizados (BOEHM, 2000), os coeficientes A e B são, inicialmente, iguais a 2.94 e 0.91 respectivamente. O expoente E é uma agregação de cinco fatores de escala, *SF* (Quadro 2), que são responsáveis pelas economias ou deseconomias de escala encontradas para projetos de software de diferentes tamanhos (DE RORE, 2008). Dito isto, o mesmo reflete o esforço aumentado necessário na medida em que aumenta o tamanho do projeto. Dessa forma, quando o expoente E é menor que 1, temos uma economia de escala. Isto significa que quando o tamanho do projeto dobrar, o efeito sobre o esforço será menos do que o dobro (BOEHM, 2000). Entretanto, quando o expoente é igual a 1, o projeto mostra uma deseconomia em escala e a duplicação do tamanho causará mais do que o dobro do esforço.

QUADRO 2 – Fatores de escala

Fator de escala	Descrição	Contexto
PREC	Precedência	O projeto é similar a outros projetos desenvolvidos anteriormente?
FLEX	Flexibilidade de desenvolvimento	Existe alguma flexibilidade em relação às exigências?
RESL	Arquitetura/Resolução de risco	Há muita atenção para a arquitetura? Os riscos foram levados em conta?
TEAM	Coesão da equipe	Existem problemas para sincronizar os diferentes *stakeholders*?
PMAT	Maturidade do processo	Qual é o nível CMM da equipe de desenvolvimento?

Fonte: DE RORE, 2008

A seleção dos fatores de escala se baseia na lógica de que eles são uma fonte significativa de variação exponencial no esforço ou variação de produtividade de um projeto (BOEHM, 2000). Cada um dos fatores de escala têm um nível de classificação, entre Muito Baixo e Extra Alto, com um peso específico.

Para os multiplicadores de esforço, *EM*, que são características do projeto que têm um efeito linear sobre o esforço de um projeto (DE RORE, 2008), devido ao nível de detalhamento entre os dois modelos apresentados, há uma diferença entre os multiplicadores que são utilizados (Quadro 3).

QUADRO 3 – Multiplicadores de esforço para os modelos preliminar e pós-arquitetura

Multiplicadores de esforço Modelo preliminar	Multiplicadores de esforço Modelo pós-arquitetura
PERS	ACAP, PCAP, PCON
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PREX	TOOL, SITE
FCIL	TOOL, SITE
SCED	SCED

Fonte: BOEHM, 2000

Para o modelo de pós-arquitetura são definidos 17 multiplicadores de esforço (Quadro 5), e, similar aos fatores de escala, há os mesmos níveis de classificação cada um com um peso determinado inicialmente utilizando os projetos na base de

dados do COCOMO II. Cada multiplicador de esforço, com exceção do Cronograma de desenvolvimento necessário, pode ser classificado para um módulo individual (DE RORE, 2008). Além disso, é possível dividi-los em várias classes:

- Fatores do produto:
  Confiabilidade de software necessária, Tamanho do banco de dados, Complexidade do produto, Desenvolvido para a reusabilidade e a Documentação compatível com as necessidades do ciclo de vida.
- Fatores da plataforma:
  Restrição de tempo de execução, Restrição de armazenamento e Volatilidade da plataforma;
- Fatores pessoais:
  Capacidade do analista, Capacidade do programador, Continuidade do pessoal, Experiência em aplicações, Experiência em plataforma e Experiência em linguagem e ferramentas
- Fatores do projeto:
  Uso de ferramentas de *software* e Desenvolvimento em vários locais

Para o modelo de projeto preliminar, todas as premissas de utilização dos multiplicadores de esforço são as mesmas. Dessa forma, para representar os dados mais simplificados, 7 multiplicadores de esforço são utilizados (Quadro 4), obtidos através da combinação dos multiplicadores de esforço do modelo de pós-arquitetura.

QUADRO 4 – Multiplicadores de esforço do modelo de projeto preliminar

Multiplicador de esforço	Descrição	Contexto
PERS	Capacidade de pessoal e mapeamento	Qual a capacidade e rotatividade da equipe?
RCPX	Confiabilidade e complexidade do produto	Qual a integridade do produto?
PDIF	Dificuldade da plataforma	Qual a integridade da plataforma?
PREX	Experiência do pessoal	Qual a experiência provida pelo projeto?
FCIL	Facilidades	Qual as facilidades associadas à equipe?

Fonte: Elaborado pelo autor com base em BOEHM, 2000

QUADRO 5 – Multiplicadores de esforço do modelo de projeto pós-arquitetura

Multiplicador de esforço	Descrição	Contexto
RELY	Confiabilidade de software necessária	Qual é o tamanho do efeito de uma falha de software?
DATA	Tamanho do banco de dados	Quantos dados de teste são necessários?
CPLX	Complexidade do produto	Quão complexo é o produto no que diz respeito a operações de controle, operação computacional, gerenciamento de dados e gerenciamento de interface de usuário?
RUSE	Desenvolvido para a reusabilidade	Os componentes são desenvolvidos para que possam ser reutilizados?
DOCU	A documentação corresponde às necessidades do ciclo de vida	Quantos dos ciclos de vida são documentados?
TIME	Restrição de tempo de execução	Existe alguma restrição no que diz respeito ao tempo de execução?
STOR	Principal restrição de armazenamento	Existe alguma restrição no que diz respeito ao espaço de armazenamento?
PVOL	Volatilidade da plataforma	Há grandes mudanças e com que frequência elas estão na plataforma?
ACAP	Capacidade do analista	Qual é a capacidade dos analistas?
PCAP	Capacidade do programador	Qual é a capacidade dos programadores?
PCON	Continuidade do pessoal	Qual é a rotatividade anual de pessoal do projeto?
APEX	Experiência em aplicações	Qual é a experiência da equipe do projeto com este tipo de aplicação?
PLEX	Experiência com a plataforma	Qual é a experiência da equipe do projeto com a plataforma?
LTEX	Linguagem e experiência com ferramentas	Qual é a experiência da equipe do projeto com as linguagens e ferramentas usadas?
TOOL	Uso de ferramentas de software	Existe alguma ferramenta de software usada para desenvolver o produto?
SITE	Desenvolvimento em vários locais	O que é a colocação do site e que suporte de comunicação está disponível?
SCED	Cronograma de desenvolvimento necessário	Qual é a restrição de cronograma imposta à equipe do projeto?

Fonte: DE RORE, 2008

2.2.1.4 Modelo de Reúso

A medição do tamanho do produto na fórmula do COCOMO II inclui todas as informações que influenciam o esforço do projeto (DE RORE, 2008). Entretanto, podemos ter código reutilizado no mesmo *software*, e o mesmo não pode ser contado da mesma forma que um código novo. Para isso, é necessário um cálculo separado para estimar o esforço necessário para integrar o código reutilizado.

O COCOMO II considera código reutilizado como caixa-preta, onde pode ser reutilizado sem compreensão ou alterações. O código adaptado é considerado caixa-branca, visto que precisa ser adaptado para se integrar com o novo código ou outros componentes reutilizáveis (SOMMERVILLE, 2016). O tamanho do código reutilizado ou adaptado precisa ser ajustado baseado no esforço adicional que é necessário para modificar o código para inclusão no produto (BOEHM, 2000). Desta forma, o KSLOC equivalente pode ser calculado através da fórmula abaixo:

$$KSLOC\ equivalente\ =\ KSLOC\ adaptado\ *\ (1\ -\ AT/100)\ *\ AAM \quad (4)$$

onde,

$$AAF\ =\ (0.4\ *\ DM)\ +\ (0.3\ *\ CM)\ +\ (0.3\ *\ IM) \quad (5)$$

$$\mathrm{AAM} = \begin{cases} \frac{\mathrm{AA}+\mathrm{AAF}\cdot(1+(0.02\cdot\mathrm{SU}\cdot\mathrm{UNFM}))}{100} & \text{for AAF} \leq 50 \\ \frac{\mathrm{AA}+\mathrm{AAF}+(\mathrm{SU}\cdot\mathrm{UNFM})}{100} & \text{for AAF} > 50 \end{cases} \quad (6)$$

O esforço necessário para reutilizar o código não começa em zero, e existem custos mesmo se o reúso não se mostrar viável. No entanto, os custos de reúso diminuem conforme aumenta o total de códigos reusados (SOMMERVILLE, 2016) Geralmente há um custo de cerca de 5% para avaliar, selecionar e assimilar o componente reutilizável (BOEHM, 2000).

Os custos fixos de compreensão e avaliação são distribuídos por mais linhas de código. Dessa forma, temos que AT representa a quantidade de código traduzido automaticamente. Para a função da quantidade de modificação, AFF, temos que, DM representa a porcentagem do projeto que foi modificada, CM representa a porcentagem de código que foi modificada e IM representa a porcentagem de esforço necessário para integrar o código adaptado ou reutilizado.

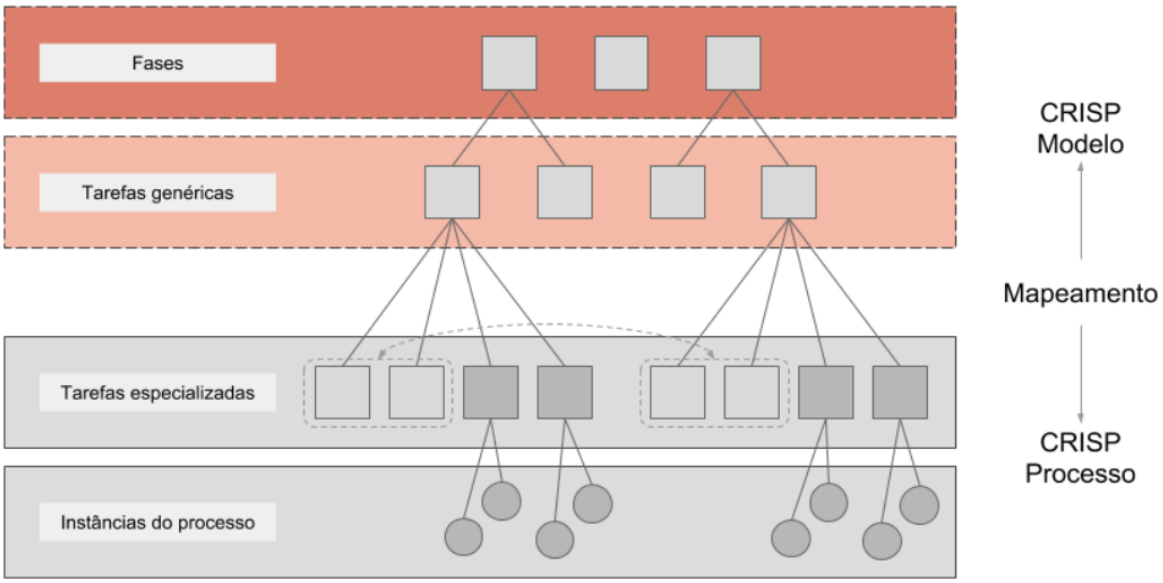
A quantidade de esforço para modificar um *software* existente depende também da compreensibilidade do software existente, SU, da relativa falta de familiaridade do programador com o *software*, UNFM e do esforço que é necessário para decidir se um módulo reutilizado é apropriado para ser usado na aplicação. Além disso, AA representa a avaliação e o incremento da assimilação do *software*.

O COCOMO II não será utilizado da maneira usual, conforme apresentado acima. O mesmo será adaptado para um contexto ágil, associado com as práticas adotadas atualmente. Esta adaptação citada, será realizada utilizando da metodologia CRISP–DM como ferramenta.

2.2.2 CRISP–DM

A metodologia CRISP–DM é descrita em termos de um modelo de processo hierárquico, compreendendo quatro níveis de abstração (WIRTH, 2000, p. 29–39), conforme apresentado através da Figura 7. A abstração apresenta-se como uma importante construção do processo de mineração de dados, de forma que é organizada em várias fases. Cada fase contém um segundo nível, representado pelas tarefas genéricas. Com o objetivo de cobrir todas as situações possíveis de mineração de dados (WIRTH, 2000, p. 29–39), as tarefas genéricas são projetadas para cobrir todos os processos e aplicações de mineração de dados. Além disso, precisam ser estáveis para desenvolvimentos ainda imprevistos. Para um terceiro nível, presente nas tarefas genéricas, é possível descrever como as ações devem ser realizadas em situações específicas através das tarefas especializadas. Para um quarto nível, a instância de processos permite um registro das ações, decisões e resultados para representar realmente aconteceu com um projeto finalizado.

FIGURA 7 – Decomposição da Metodologia CRISP–DM em quatro níveis para mineração de dados



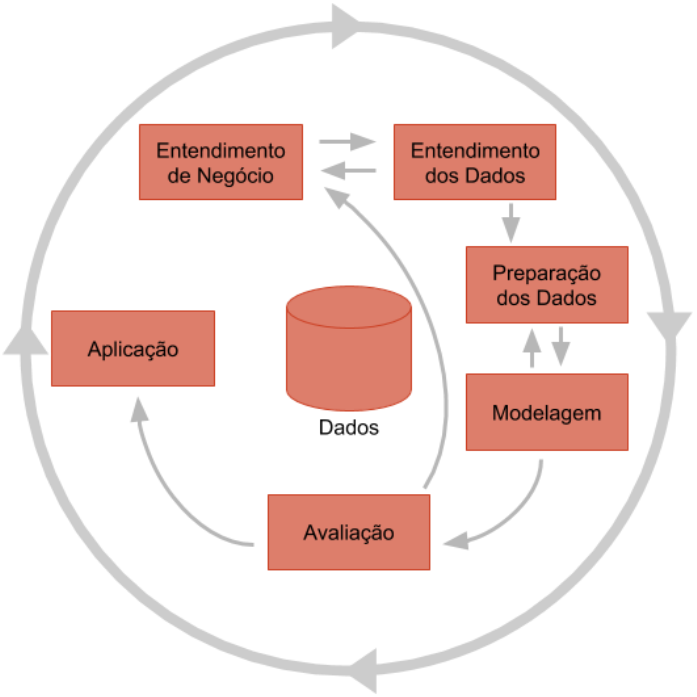
Fonte: WIRTH, 2000, p. 29–39

Devido o CRISP–DM fornecer uma visão geral de um projeto de mineração em seis fases (Figura 8), muitas das tarefas podem ser executadas em uma ordem diferente, sendo necessário voltar e repetir determinadas ações (WIRTH, 2000, p. 29–39). Para capturar todas as possibilidades exigiria um modelo de processo excessivamente complexo e os benefícios esperados seriam muito baixos (WIRTH, 2000, p. 29–39).

2.2.2.1 Entendimento do Negócio

Esta fase inicial se concentra na compreensão dos objetivos e exigências do projeto a partir de uma perspectiva de negócios, configurando-se como uma das fases mais importantes de qualquer projeto de mineração de dados (SHEARER, 2000, v. 5, n. 4, p. 13–22). A fim de entender plenamente o negócio envolvido, geralmente o conhecimento é convertido em uma definição do problema de mineração de dados e, em seguida, é desenvolvido um plano preliminar projetado para atingir os objetivos. Para isso, analistas de dados devem descobrir o principal objetivo, bem como as questões relacionadas que o negócio gostaria de abordar (SHEARER, 2000, v. 5, n. 4, p. 13–22).

FIGURA 8 – Fases do atual modelo do processo CRISP-DM para mineração de dados



Fonte: WIRTH, 2000, p. 29–39

2.2.2.2 Entendimento dos Dados

Nesta fase, os dados iniciais são coletados e os analistas de dados identificam problemas de qualidade, descobrem ideias iniciais sobre os dados, identificam subconjuntos interessantes e fazem hipóteses sobre informações, em grande maioria, implícitas. Além disso, há uma ligação entre a compreensão do negócio e a compreensão dos dados (WIRTH, 2000, p. 29–39). Dessa forma, é importante entender os dados disponíveis para desenvolver problemas de mineração de dados e planejamento de projetos.

2.2.2.3 Preparação dos Dados

Todas as ações relacionadas com a construção do *dataset* final estão presentes nesta fase. Com foco nas metas de mineração de dados, é necessário selecionar os dados através da relevância e das restrições de qualidade e técnicas. Durante esta fase, há uma limpeza, formatação dos dados e remoção de caracteres ilegais. Além disso, pode haver a criação de novos atributos ou registros para os dados para facilitar o processo ou algoritmo de modelagem. Outro passo importante

que ocorre nesta fase é a integração de dados de múltiplas fontes, combinando informações para criar novos registros. Além disso, novos valores podem ser calculados através do resumo de informações de múltiplos registros e/ou tabelas.

2.2.2.4 Modelagem

Nesta fase, várias técnicas de modelagem são selecionadas e aplicadas, e seus parâmetros são calibrados (WIRTH, 2000, p. 29–39). Com etapas como seleção da técnica de modelagem, a geração do projeto de teste, a criação de modelos e a avaliação dos modelos requerem uma construção específica sobre os dados, então voltar à preparação dos dados fase pode ser necessário (SHEARER, 2000, v. 5, n. 4, p. 13–22).

2.2.2.5 Avaliação

Com a escolha da técnica de modelagem, nesta etapa é realizada uma revisão para avaliar mais profundamente o modelo, rever as fases executadas para a construção, a fim de garantir os objetivos de negócios envolvidos. Os principais pontos desta etapa é determinar se existe algum ponto importante da regra de negócio que não tenha sido suficientemente considerada (WIRTH, 2000, p. 29–39) e determinar os próximos passos.

2.2.2.6 Aplicação

Com todas as fases anteriores finalizadas, esta tem o objetivo de organizar e apresentar todo o conhecimento adquirido para utilização dos clientes envolvidos. Dependendo dos requisitos, a fase de implantação pode ser simples, para gerar relatórios ou complexa, para implementar um processo de mineração de dados escalável. As principais etapas são a implantação do plano, o monitoramento e a manutenção do plano, a produção do relatório final e a revisão do projeto (SHEARER, 2000, v. 5, n. 4, p. 13–22)

3 MATERIAIS E MÉTODOS

Com o objetivo de prover uma solução orientada através dos dados, o presente trabalho conduzirá a metodologia inspirada em processos de mineração de dados. As próximas seções detalham os materiais e métodos relacionados com o desenvolvimento do presente trabalho.

3.1 MATERIAIS

3.1.1 Objetivo de Pesquisa

Para utilização e análise da criação de um processo automatizado de estimativa, serão utilizados os dados referente a produção, através da contribuição de novas alterações de código por parte de integrantes do time ágil. Os dados citados serão extraídos de plataformas diferentes, onde, respectivamente, estão as informações quanto aos códigos desenvolvidos e o gerenciamento das estórias e o valor agregado à entrega do que foi finalizado.

3.1.2 Ferramentas Utilizadas

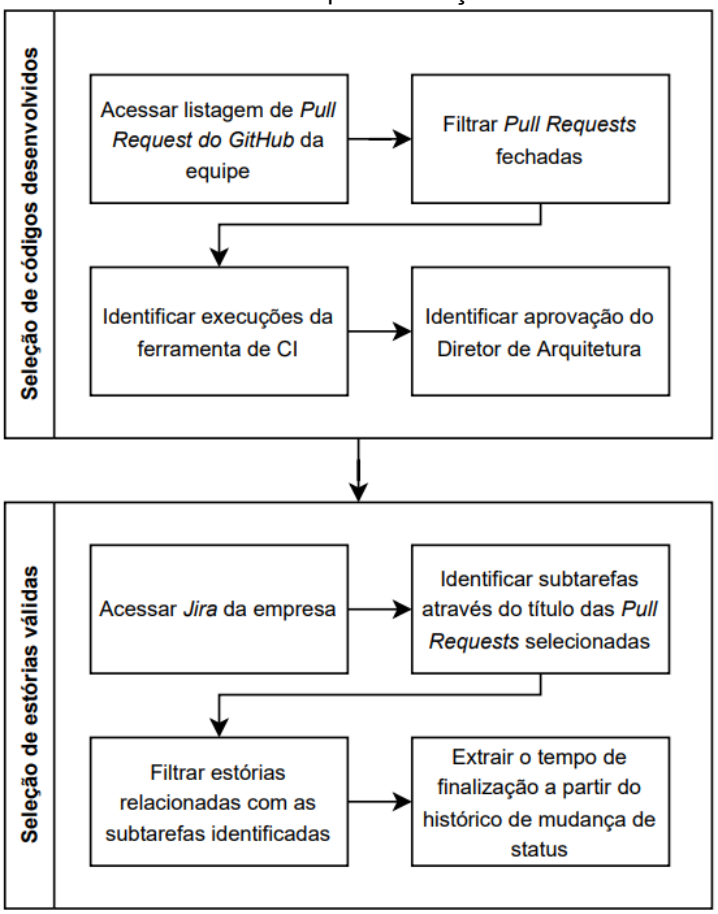
Será utilizado o modelo de estimativa COCOMO II como fator comparativo com dados reais coletados de plataformas que são gerenciadas pelo time ágil. Os dados foram extraídos através de um programa desenvolvido em *Python* e executado através da ferramenta *Google Collaboratory*. Para consumo dos dados citados, foi integrado ao programa as *APIs* do *Github* e *JIRA*.

3.2 MÉTODOS

Para o desenvolvimento do método automatizado para obtenção de dados estruturados foi utilizado APIs do *GitHub* e *Jira*, para extração de informações do código contribuído pelo time e informações da estória relacionada com o que foi

desenvolvido, respectivamente (Figura 11). Ambas as plataformas são utilizadas pela empresa como um todo, de forma que, para adaptar essa abordagem para outro time seria necessário apenas alterar os filtros relacionados.

FIGURA 11 – Fluxo para obtenção dos dados



Fonte: Elaborado pelo autor

E, para um procedimento mais concreto e maduro, independente tanto do setor industrial quanto da tecnologia utilizada (WIRTH, 2000, p. 29–39), e que inicia-se com o estudo do negócio, foi adotada a metodologia CRISP–DM, em detrimento de outras como o SEMMA e PMML. Com uma taxa de 42% de preferência entre os profissionais (LAUREANO *et al.*, 2014, v. 0, n. 13, p. 83–98), o CRISP–DM ou *Cross Industry Standard Process for Data Mining*, é uma abordagem abrangente para mineração de dados que fornece um plano para conduzir projetos de mineração de dados (SHEARER, 2000, v. 5, n. 4, p. 13–22).

4 ACOMPANHAMENTO DA PRODUTIVIDADE DE UM TIME ÁGIL ATRAVÉS DO COCOMO II

O presente capítulo tem como objetivo expor as decisões e processos estabelecidos para utilização de um modelo de estimativa para acompanhamento da produtividade. Todos os métodos envolvidos foram realizados em uma empresa privada com uma grande adoção interna quanto às metodologias ágeis. Especificamente o time ágil exposto utiliza do método Kanban para realização do desenvolvimento de *software*, contando com processos menos opinativos quando comparados com SCRUM ou XP. Além disso, o time conta com 99% dos integrantes recém-absorvidos, todos com menos de 1 ano. Diante desse contexto, o acompanhamento da produtividade torna-se bastante desafiador, principalmente quando realizado manualmente, de forma que, pode ser um processo subjetivo e bastante volátil quando há questões com prazos e qualidade nas entregas.

Após contato com gerentes e líderes técnicos do time, chegou-se à conclusão das necessidades associadas à uma solução para realizar a mensuração e acompanhamento. Primeiramente, era necessário que o tempo e o esforço necessários para coletar os dados fossem mínimos para não gerar desperdícios de tempo ou despesas. Dessa forma, seriam preferidas técnicas que tivessem a oportunidade de automatizar o processo de medição. Além disso, a medição deveria ser feita em momentos específicos durante a execução de um projeto pelo método Kanban. Todo o desenvolvimento ocorre na fase de *Downstream*, portanto era importante encontrar uma alusão com o RUP, onde grande parte dos modelos de estimativas operam sobre esta metodologia. Com isto, esta abordagem poderia permitir um benchmarking com outros times, com a precisão avaliada com base no portfólio e não por projeto.

Dessa forma, como solução para atender as expectativas acima e o contexto envolvido, foi considerada a utilização do COCOMO II, um modelo de estimativa, como medida de produtividade para realizar o acompanhamento. Com a utilização

das linhas de código como medida de tamanho do *software*, os mesmos podem ser contados automaticamente. Além disso, com a estabilização de um ambiente que possa descartar os paradoxos envolvidos com os SLOCs é possível evitar que conclusões erradas sejam tiradas dos resultados.

4.1 ENTENDIMENTO DO NEGÓCIO: COCOMO II COMO MEDIDA DE PRODUTIVIDADE

A fim de trazer uma padronização e modelação ao processo de mensuração da produtividade, o COCOMO II será utilizado como medida de produtividade ao invés da análise simples dos dados históricos. Através desta abordagem, poderá ser possível estimar o trabalho necessário para desenvolver qualquer projeto, além de definir orçamentos, cronogramas e *tradeoff* envolvidos. Para isto, será realizada uma comparação entre a estimativa de esforço com o esforço real que foi registrado para determinado desenvolvimento de *software*. Desta forma, por exemplo, um projeto pode ser determinado como menos produtivo se houver mais esforço do que foi estimado pelo modelo.

Entretanto, mesmo que seja apresentado como uma alternativa dos dados históricos, os mesmos ainda são importantes para o processo, principalmente quanto ao acúmulo para uma análise através de uma perspectiva retroativa. Dessa forma, o resultado do COCOMO II será utilizado como fator de comparação e os dados históricos serão importantes no futuro para um aperfeiçoamento e refinamento. Portanto, o método de medição que deve ser automatizado, precisa considerar a cada nova mensuração, os dados históricos para gerar os resultados do modelo e assim trazer a resposta mais assertiva e contextualizada possível.

Com a definição do COCOMO II como modelo de estimativa escolhido e com a adoção do mesmo como medida de produtividade, é importante definir pontos críticos para serem solucionados a fim concluir se uma medição com o modelo COCOMO II é válida. Este modelo de estimativa utiliza linhas de código e vários fatores de escala e multiplicadores de esforço para estimar o esforço necessário, para uma avaliação correta da produtividade é importante avaliar os mesmos

atributos que o modelo COCOMO II analisa. Além disso, é necessário avaliar em qual momento é o mais válido para ser feito as comparações, dentro dos processos já adotados pela equipe ágil. Dessa forma, temos alguns pontos importantes para considerar, dentre eles, uma contagem correta das linhas de código e uma relação correta entre as informações de esforço e tamanho do *software* mensuradas através do método Kanban.

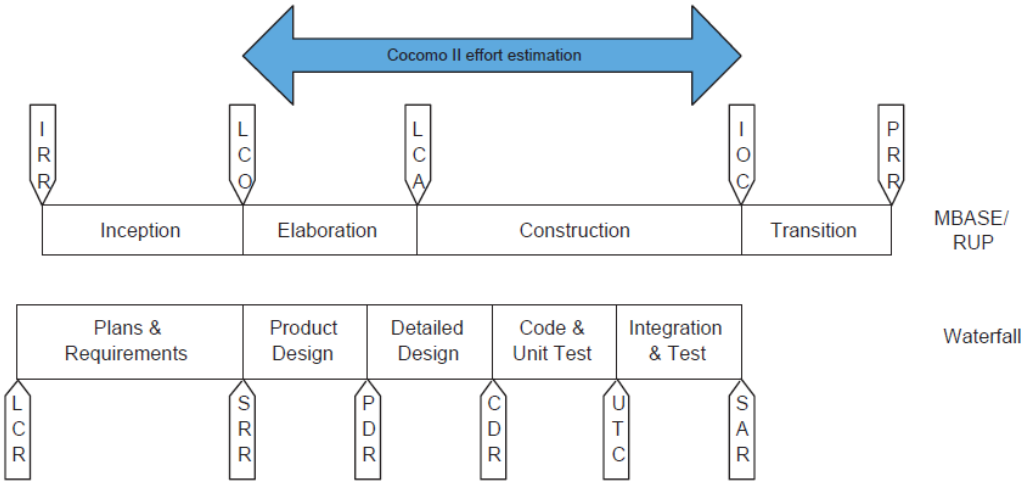
Para uma utilização cada vez mais assertiva quanto a comparação dos resultados do COCOMO II para mensuração da produtividade, precisa-se uma atenção maior quanto aos atributos do modelo, para serem os mais corretos e contextualizados possíveis. Portanto, para a mensuração do tamanho do *software*, através de KSLOC, é necessário seguir os passos e cálculos já estabelecidos, entretanto com o foco na automatização do processo.

Para uma contextualização quanto ao time ágil presente na empresa que foi realizado o acompanhamento, o mesmo desenvolve a partir de uma iniciativa low-code, onde todo o back-end já existe. A equipe é composta, em grande parte, por desenvolvedores front-end, e dessa forma, duas linguagens de programação são fortemente atuantes no processo de criação de um produto. Associado a este contexto, conforme já apresentado, há paradoxos inerentes à utilização de SLOCs como medidas de tamanho (JONES, 1986), que serão discutidos posteriormente.

4.2 ENTENDIMENTO DOS DADOS: APLICAÇÃO DO COCOMO II NO MÉTODO KANBAN

Através do método ágil Kanban, o time, onde será aplicado o modelo, segue o processo para desenvolvimento de projetos, o que torna necessário desenvolver uma lógica para adaptação neste processo. Uma importante característica e diferencial do COCOMO II, é sua modelagem associada com processos de desenvolvimento de produtos, sendo crucial para o registro do tempo, como carga de trabalho e esforço. Entretanto, o modelo, de acordo com a época que foi desenvolvido, adapta-se com metodologias que não possuem mais uma grande adesão (Figura 10), como RUP e modelo em cascata.

FIGURA 10 – Utilização do COCOMO nos processos RUP e modelo em cascata

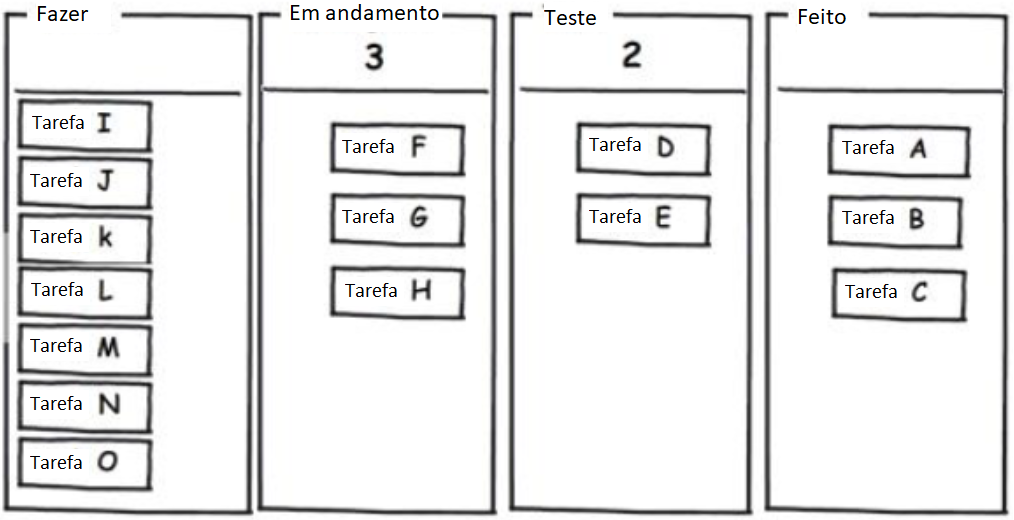


Fonte: DE RORE, 2008

Devido a dinâmica e a velocidade quanto às entregas do time, o modelo será utilizado numa perspectiva menor, através da análise de estórias ao invés de avaliar o projeto inteiro, sendo este contínuo, de acordo com as práticas da metodologia ágil. E, para desenvolvimento das mesmas, o método Kanban divide todas as etapas em duas grandes fases, conforme já abordado, *Upstream* e *Downstream*.

De acordo com COCOMO II (BOEHM, 2000), dentro da visão RUP ou em cascata, os planos e a fase de requisitos não devem ser contados como parte do esforço de desenvolvimento. Em alusão, estas etapas podem ser contidas dentro do *Upstream*, com isto, o mesmo não será contabilizado. Para uma relação entre esforço e tamanho do *software* criado pelos desenvolvedores, é necessário realizar uma análise mais detalhada quanto ao *Downstream*, fase esta onde estão todas as etapas de desenvolvimento e considerar apenas as etapas de desenvolvimento mais intenso.

FIGURA 11 – Exemplo de um quadro Kanban na fase de *Downstream*



Fonte: KIROVSKA; KOCESKI, 2015, v. 3, n. 3, p. 25–34

Desta forma, a partir das etapas do *Downstream* (Figura 11), para contexto mais assertivo quanto aos atributos de esforço e tempo associados, em alusão com as metodologias compatíveis com o COCOMO, será considerado a transição das estórias entre as colunas, ou etapas do processo. Ou seja, para o registro do tempo, será considerado o tempo que a estória ficou na etapa, “Em Progresso” e transitou para a fase “Pronta para testar”, ou “Em teste”.

4.2.1 Compreensão dos dados

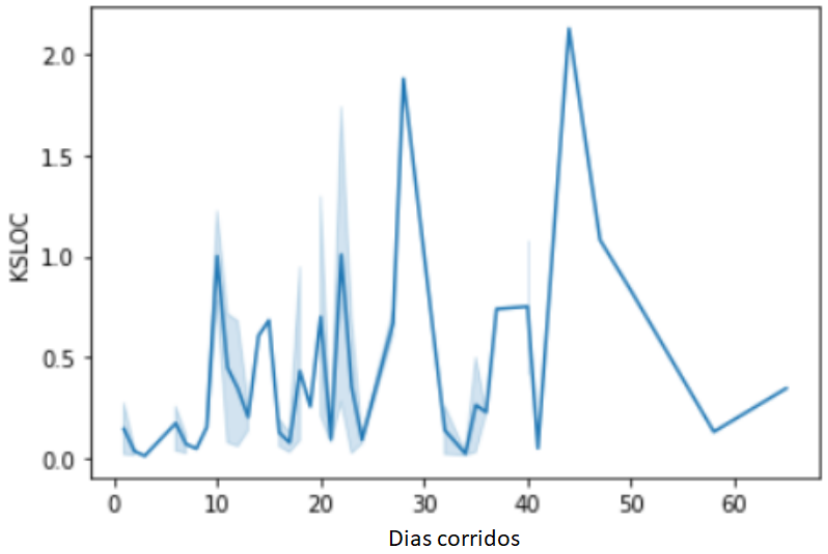
Através de um ponto de vista mais amplo, é possível atribuir ao conjunto de dados a orientação pelas estórias que foram desenvolvidas pela equipe. Com o identificador, foi possível filtrar todos os códigos que foram responsáveis pela conclusão da funcionalidade ou correção. Dessa forma, é possível ter uma relação direta entre o tempo que a estória passou para a fase de testagem, com a quantidade de código que foi produzido, assim como, extrair novas informações de códigos produzidos por dia, quantidade de pessoas e o prazo exato.

4.2.1.1 Análise Descritiva Dos Dados

Para uma visualização mais clara e assertiva quanto a base de dados, a seguir serão apresentadas algumas análises através das relações entre os atributos

presentes. Com uma compreensão facilitada e uma maior evidência quanto às tendências dos dados, pode ser possível identificar problemas associados e melhores resultados.

FIGURA 12 – Gráfico de KSLOCs por dias passados



Fonte: Elaborado pelo autor

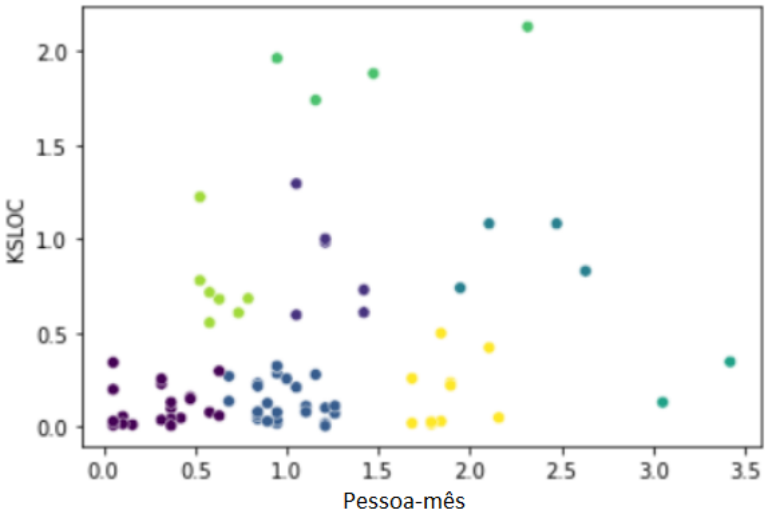
A Figura 12 representa a plotagem, utilizado o gráfico de linha, KSLOC versus dias passados. Esta representação extrai a informação de distribuição de linhas de código a medidas que é desprendido mais dias para desenvolver uma estoria. Dessa forma, pelo estudo realizado, percebe-se, por exemplo, que entre 30 e 40 dias passados, tem uma variação bastante significativa. Através disso, pode indicar uma maior dificuldade quanto à resolução da estória ou ao processo que não foi bem sucedido.

A Figura 13 representa a plotagem de uma clusterização realizada pela técnica K-Means, a partir da biblioteca *Scikit Learning*, no *Python*, utilizando Pessoa-mês versus KSLOC. Foi utilizado os valores padrões, com a definição de 8 *clusters* a serem formados, bem como os centróides a serem gerados. Além disso, também com valores padrões foram realizadas 300 interações.

Dessa forma, através da análise, foi permitido a visualização de possíveis clusters quanto à relação envolvida entre os KSLOC pelo esforço atribuído. Dessa forma, trouxe maior clareza quanto a relação entre pessoa-mês, ou dia, com as linhas de código, porém sem uma crescente 1:1. Por exemplo, os valores de

pessoa-mês entre 0.5 possuem em torno de 3 clusters, sendo 2 deles bem próximos quanto ao valor de KSLOC. Entretanto, para um resultado mais satisfatório, para a contabilização das linhas de código, foi necessário mitigar alguns paradoxos relacionados.

FIGURA 13 – Clusters auto gerados a partir dos dados



Fonte: Elaborado pelo autor

4.2.1.2 Paradoxos Associados Com A Contagem Das Linhas De Código

Visto que o objetivo é determinar um processo independente do projeto que esteja sendo executado, uma solução parcial pode ser suficiente, desde que os paradoxos sejam, em grande parte, resolvidos. Os seguintes paradoxos e as escolhas tomadas para tentar solucioná-los são discutidos a seguir:

1. **Uso de diferentes linguagens de programação**

O cotidiano de desenvolvimento da equipe, conforme abordado, é relacionado ao front-end, dessa forma, foi decidido considerar somente as linhas de código desenvolvidas com *Javascript*. Outras linguagens são utilizadas, como *JSON*, *CSS/SASS* e *HTML*, mas representam uma minoria de código novo e portanto serão desconsideradas. Além disso, a linguagem escolhida traz um maior valor agregado

em comparação com as outras, que não é possível associar as regras de negócio e servem mais como apoio.

2. **Novas linhas de código contra linhas modificadas e conflitos de entrega da mesma parte do código**

Para o processo correto do COCOMO II, diante de uma parte do código, como módulo, componente ou pacote, só deve ser considerado as linhas que de fato foram modificadas ou criadas. O mesmo vale para os conflitos de mudança de uma mesma parte, é necessário distinguir entre código novo e modificado. Para controlar toda essa dinâmica, a empresa e o time adotaram o *Git* e *Github* para gerenciamento de mudanças e versões do código. Dessa forma, é possível, até mesmo, recuperar informações retroativas para identificação mais assertiva quanto às mudanças realizadas por cada desenvolvedor. Além disso, a utilização dessas ferramentas foram determinantes para a escolha do modelo, que utiliza LOCs, pois foi possível realizar a automatização de forma mais orgânica.

3. **Diferentes estilos de programação**

Embora alguns dos paradoxos com linhas de código já tenham sido contrabalançados pelo uso de fatores de conversão e fatores de redução para o código gerado (DE RORE *et al*., 2008), quando relacionados ao estilo de código manual, ainda há um paradoxo para ser abordado. Dessa forma, a principal preocupação envolvida está na forma que um desenvolvedor pode estruturar seu código. Alguém que escreva de forma desestruturada, na visão de linhas de código, pode parecer que foi mais produtivo que outro que desenvolveu de forma mais sofisticada. Para solucionar esse problema, o time incorpora uma base de boas práticas que são ensinadas durante o ingresso na empresa, de forma que os estilos de desenvolvimento convergem para o estilo utilizado por todos. Além disso, o time possui práticas de programação pareada, absorvido principalmente das metodologias ágeis, e revisão de código, onde o líder ou alguém com senioridade maior, realiza a identificação de melhorias e ajustes, de acordo com os padrões.

Em outra perspectiva, para solucionar problemas de código morto, o time, também através das práticas de metodologias ágeis, aplica sessões de refatoração do código. Dessa forma, é possível evitar que o projeto possua vários código mortos e aparente que foi mais produtivo em comparação com outros.

4. Novas linhas de código contra linhas reulizadas

Mensurar o reúso de trechos de código é uma tarefa difícil. Assim como código modificado, o reutilizado também precisa ser mensurado de forma diferente se comparado com um código novo. Entretanto, mesmo com ferramentas como *Git*, a mensuração das linhas de código são comprometidas, principalmente no desenvolvimento front-end, onde módulos, componentes e bibliotecas são reutilizadas praticamente em todas as partes do código. Dessa forma, seguindo o processo do COCOMO II, onde o esforço envolvido e a mensuração de um código reutilizado são diferentes, é possível utilizar do modelo de reuso do COCOMO, considerando que nem todos os critérios da fórmula são aplicáveis, temos a fórmula:

$$KSLOC\ equivalente\ =\ KSLOC\ adaptado\ *\ (AA\ +\ 0.3\ *\ IM)/100 \qquad (7)$$

Foi necessário uma simplificação quanto a fórmula original, visto que não está sendo considerado códigos traduzido ($AT = 0$) ou códigos modificados ($DM = 0$, $CM = 0$ e $SU = 0$). Além disso, foi adotado valores padrões para os parâmetros, a partir da análise da maturidade atual da equipe quanto a criação e utilização de componentes, serviços e módulos para reutilização durante todo o projeto. Desta forma, a avaliação e o incremento da assimilação, $AA = 2$, onde, em contexto com a equipe, significa que é necessário uma documentação básica e uma busca para determinar se o componente é apropriado para o projeto. Além disso, $IM = 20$, referente a integração necessária para um *software* adaptado, contextualizado aqui como um componente, serviço ou módulo. Em outras palavras, 20% do esforço é necessário para integrar o *software* adaptado no programa. Desta forma, o código reutilizado representa 8% do número de linhas que são de fato reutilizadas.

Finalmente, para utilização do resultado obtido acima dentro dos cálculos de mensuração da forma mais aprimorada possível seria necessário bastante recurso

computacional. Seria necessário um banco de dados com informações de cada componente que pudesse ser um código para ser reutilizado e o cálculo ser aplicado para cada consulta. Desta forma, como alternativa mais simples, foi assumido que este resultado representa um fator de redução constante que deve ser aplicado a cada contagem de linhas de código.

4.3 PREPARAÇÃO DOS DADOS: FATORES DE ESCALA E MULTIPLICADORES DE ESFORÇO

Conforme comentado anteriormente, o COCOMO II foi utilizado para realizar a mensuração de cada estória, entretanto, como todas as supracitadas estórias fazem parte de um mesmo projeto, foi realizado a análise uma única vez e aplicado os mesmos fatores de escala e multiplicadores de esforço para cada mensuração. Com isto, foi realizada uma análise juntamente com os gerentes e líderes técnicos do time para classificar cada um dos pontos entre Muito baixo e Extra alto, o que seria representado em fatores durante o cálculo do modelo. Dessa forma, as respostas obtidas e analisadas para o contexto do time estão expressas nos quadros 6 e 7.

QUADRO 6 – Fatores de escala utilizados para mensurações

Fator de Escala	Classificação	Contexto
PREC	Alto	O produto é geralmente familiar com outros
FLEX	Muito Alto	Alguma conformidade com produto
RESL	Baixo	Há algum nível de arquitetura e gerenciamento de riscos
TEAM	Alto	Os *stakeholder* são altamente cooperativos
PMAT	Alto	SW-CMM Nível 3

Fonte: Elaborado pelo autor

QUADRO 7 – Multiplicadores de esforço utilizados para mensurações

Multiplicador de esforço	Classificação	Contexto
RELY	Baixo	Perdas baixas e facilmente recuperáveis.
DATA	Baixo	Teste de bytes DB/Pgm SLOC < 10.
CPLX	Nominal	Baixa complexidade, procedimentos padrões.
RUSE	Nominal	Através do projeto.
DOCU	Baixo	Algumas necessidades do ciclo de vida do usuário precisam ser descobertas.
TIME	Nominal	≤ Uso de 50% do tempo de execução disponível.
STOR	Nominal	≤ Utilização de 50% do armazenamento disponível.
PVOL	Alto	Mudanças maiores a cada 2 meses; mudanças menores a cada 1 semana.
ACAP	Baixo	35º percentil.
PCAP	Baixo	35º percentil.
PCON	Nominal	12% / ano.
APEX	Nominal	1 ano.
PLEX	Nominal	1 ano.
LTEX	Nominal	1 ano.
TOOL	Alto	ferramentas fortes, maduras, de ciclo de vida, moderadamente integradas.
SITE	Extra alto	Totalmente colocado e com interações online.
SCED	Baixo	85% do nominal.

Fonte: Elaborado pelo autor

4.4 MODELAGEM: MÉTODO DE MEDIÇÃO ORIENTADA PELOS DADOS

Conforme abordado em seções anteriores, para um melhor direcionamento e estruturação do dados, todo o processo a seguir foi inspirado na metodologia CRISP–DM, com um foco maior no entendimento e tratamento de dados a serem aplicados no modelo COCOMO II. Para cada uma das fases relacionadas a metodologia foi realizado uma análise quanto a sua relevância e utilidade para ser apresentada no presente trabalho. Dessa forma, as fases de Compreensão do negócio, Avaliação e Implementação foram desconsideradas, visto que o conteúdo e o objetivo da informação proposta estão diluídos por todo este trabalho. Com isto, para uma visão orientada aos dados serão abordadas etapas inspiradas nas fases de Compreensão dos dados, Preparação dos dados e Modelagem.

Além disso, alinhado com a necessidade de um processo automatizado, as supracitadas etapas foram elaboradas através de um software desenvolvido na linguagem de programação *Python* e executada pelo *Google Collaboratory*. A escolha das tecnologias envolvidas estão relacionadas com a grande adoção por parte da comunidade acadêmica e, através do mercado, as mesmas são utilizadas para desenvolvimento de provas de conceito, sendo este o intuito do uso.

4.4.1 Obtenção dos Dados Iniciais

Através do repositório utilizado para envio de *Pull Requests* foi possível obter todos os códigos desenvolvidos, porém foi considerado apenas aqueles que de fato foram integrados ao código-fonte do produto. A identificação destes foi através do reconhecimento de execuções da ferramenta de entrega contínua, ou *Continuous Delivery* e da aprovação por parte do Diretor de Arquitetura da empresa. Com isto, cada *Pull Request* possuía um identificador único relacionado com subtarefas de uma estória e através dessa informação foi possível integrar os dados presentes com o *Jira*, identificando assim as estórias que foram desenvolvidas. Dessa forma, pela plataforma e com os registros associados, foi possível agregar os dados de quanto tempo a estória foi finalizada, ou seja, passou para a fase de testagem,

através da mudança de status registrada retroativamente. Entretanto, para obtenção de informações mais detalhadas, foi necessário descartar após identificação do *Pull Request, as* estórias: que estivessem em andamento, com dados incompletos quanto ao início ou fim do desenvolvimento, tarefas técnicas e bugs. Ou seja, foi considerado apenas estórias referente a funcionalidades ou correções que trouxessem valor para o cliente.

Com isto, foi extraída uma base de dados em .csv, que agrega informações relevantes das duas plataformas para que seja convertida em atributos para utilizar em conjunto com o modelo COCOMO II. Além disso, para uma análise quantitativa mais aperfeiçoada, foram adicionados atributos, como dias passados, pessoa-hora e pessoa-mês, a partir das informações extraídas de mudança de status das estórias no *Jira*. Para obtenção destes últimos dois novos atributos foi levado em consideração a mesma abordagem utilizada pelo COCOMO II, com um valor nominal de 152 horas por pessoa-mês. Para identificação de fins de semana e feriados, para um cálculo mais assertivo, foi utilizado uma API integrada com o *Python*, onde foi calculada junto com o histórico de mudança de status supracitado para identificar o fim do desenvolvimento. Com isto, todos os novos atributos puderam ser mais aperfeiçoados quanto aos dias úteis e até mesmo as 8 horas úteis, consideradas para o desenvolvimento. Dessa forma, foi possível obter uma base de dados com 5 atributos (Quadro 8), sendo o primeiro deles o identificador da estória presente no Jira; o segundo são referências de commits em SHA para identificação via Git, dos trechos de código que foram adicionados ou alterados; os atribuídos a seguir são os mesmos abordados acima, dias passados, pessoa-hora e pessoa-mês.

QUADRO 8 – Dicionário parcial dos dados

Atributo	Tipo	Tamanho (bytes)
Issue Key	Texto	10
Commit Ranges	Texto	100
Days Spent	Número	4
Person-Hour	Número	4
Person-Month	Número	4

Fonte: Elaborado pelo autor

4.4.2 Preparação do Atributo Comparativo

Como última etapa para a formação de uma base de dados compatível com as entradas e resultados do modelo COCOMO II, é necessário realizar a contagem de linhas de código. Para isto, a partir do atributo Commit Range, com auxílio das ferramentas *Git* e *cloc*, foi possível extrair a informação de SLOC para cada estória. E, através do *commit SHA* foi possível identificar os códigos modificados, com o projeto analisado localmente. Para cada *commit* pode ser gerado dados de diferenças, ou *diff*. A partir disso, o *cloc* atuou para gerar um relatório em .json que distinguiu entre as linguagens que foram utilizadas, comentários, linhas em brancos, além de diferenciar entre código novo e modificado.

Dessa forma, através do *Python*, foi possível extrair de cada linha, onde o *cloc* gerou um o relatório, os KSLOC corretamente. Primeiro removendo comentários e linhas em branco do cálculo total e por fim, aplicando o modelo de reúso e o fator de diferença, abordados anteriormente para encontrar os KSLOC equivalentes. Com isso, é substituído o atributo Commit Ranges por KSLOC.

4.5 AVALIAÇÃO: APLICAÇÃO DO COCOMO II

Finalmente, com a base de dados completa e o atributo que representa o esforço real de cada estória presente, através do *Python*, foi possível aplicar o modelo COCOMO II. Através dos fatores de escala, multiplicadores de esforço e variáveis iniciais e o conjunto de dados atribuídos como entrada do modelo, foi possível criar uma nova coluna, com o resultado em pessoa-mês gerado pelo modelo (Quadro 9). Dessa forma, é possível realizar as comparações entre o esforço real e o estimado, utilizando o resultado de pessoa-mês por KSLOC como medida.

QUADRO 9 – Dicionário final dos dados

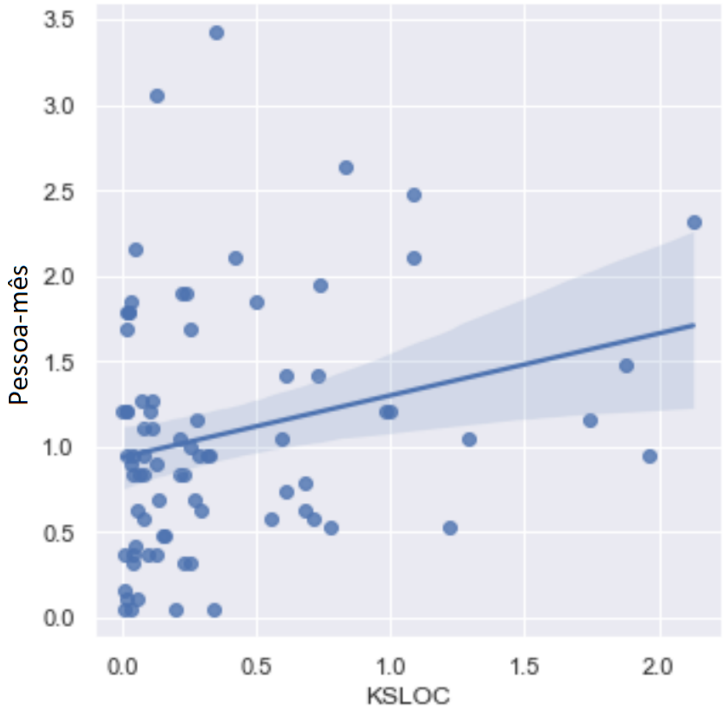
Atributo	Tipo	Tamanho (bytes)
Issue Key	Texto	10
KSLOC	Número	40
Days Spent	Número	4
Person-Hour	Número	4
Estimated Person-Month	Número	4
Real Person-Month	Número	4

Fonte: Elaborado pelo autor

4.6 APLICAÇÃO: RESULTADOS

Portanto, através dos dados obtidos e com a aplicação do modelo COCOMO II, onde resultou em uma nova coluna de pessoa-mês estimada, foi possível realizar comparações com a relação entre KSLOC e pessoa-mês. O mesmo pode ser associado à ideia de produtividade, visto que pode ser considerado como quantidade de produção dividida pelo esforço necessário. Para o contexto de desenvolvimento de software, o esforço pode ser medido como horas de trabalho gastas, onde a medida de pessoa-mês se relaciona também com as pessoas envolvidas. Dessa forma, a partir da Figura 14, é plotado a representação real dos dados extraídos desde toda a existência do time e com as estórias válidas, contabilizando 79.

FIGURA 14 – Gráfico de pessoa-mês por KSLOCs extraídos de dados reais



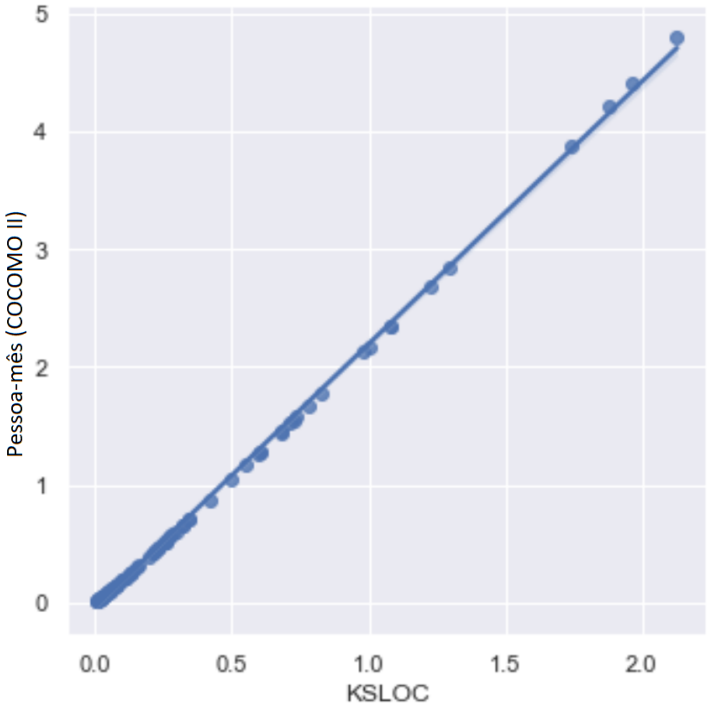
Fonte: Elaborado pelo autor

Através desta representação real, é possível identificar o perfil de demanda que a equipe ágil trata durante o desenvolvimento, visto que a grande maioria das estórias plotadas ficam em torno de 0.0 e 0.5 KSLOCs. Além disso, é possível avaliar o desempenho do time como um time constante, visto que o crescimento do

atributo pessoa-mês, através da linha azul, é uma linha sem variações bruscas, mesmo que haja algumas estimativas, através da região azulada, quanto a variação.

Entretanto, o perfil identificado pela quantidade de estórias em uma única região trouxe o problema de haver poucos dados quanto a KSLOCs maiores. Mesmo que seja considerado um desempenho constante, através de mais informações seria possível um embasamento mais concreto. Visto que o presente trabalho foi realizado com uma equipe relativamente nova, composta de colaboradores também recentemente associados ao time, relacionado com apenas 1 projeto que estava em andamento, não foi possível gerar uma base de dados massiva. Dessa forma, para os resultados obtidos pela utilização do modelo COCOMO II, através da Figura 15, foi possível representar os valores estimados para pessoa-mês.

FIGURA 15 – Gráfico de pessoa-mês por KSLOCs extraídos de dados estimados



Fonte: Elaborado pelo autor

Com as estórias plotadas, foi possível obter uma representação pouco satisfatória e conclusiva quanto ao que foi estimado pelo modelo para acompanhar os dados reais. Este resultado era provável devido às condições já expostas quanto à quantidade de dados analisados. Além disso, visto que o COCOMO II é um

modelo de estimativa para projetos complexos e de grandes companhias, o mesmo espera-se que seja utilizado com *softwares* com SLOCs maiores que 2000 ou 2 KSLOCs. Dessa forma, visto que a diferença entre os dois resultados são bem expressivas e pouco conclusiva, não foi realizado uma comparação visual para uma análise quanto à previsibilidade do modelo para uma futuro, através de uma margem de erro.

5 CONCLUSÃO

Dessa forma, através do presente trabalho, foi possível concluir que é possível a utilização do COCOMO II junto com processos automatizados de extração de dados associado a um time ágil. Entretanto, não foi possível obter resultados satisfatórios quanto à estimativa, visto que, pelo contexto onde foi realizado a análise, não houve uma quantidade significativa de dados.

Com a automatização do processo de extração dos dados proposta, e através das premissas assumidas quanto à paradoxos envolvidos, foi possível estabelecer um padrão para utilização em outros times, até mesmo da mesma empresa. A padronização estabelecida, refletida pelo software desenvolvido em *Python* pode ser ajustada para utilização do *Jira* e *Github* de outros contextos para que possa gerar os dados similares ao que foram obtidos. Além disso, com a similaridade das etapas e fases entre os métodos ágeis, pode ser possível replicar esta abordagem, que foi desenvolvida para o método ágil, para outros processos, como o SCRUM.

Portanto, apesar dos resultados obtidos, pouco satisfatórios, houve um grande entendimento sobre os processos necessários, principalmente para obter os dados que são utilizados como entradas do COCOMO II, bem como calcular resultados na mesma medida utilizados pelo modelo.

5.1 TRABALHOS FUTUROS

Com o contexto apresentado acima, o presente trabalho pode ser uma importante ferramenta para uma melhoria contínua do processo de utilização de modelos de estimativa para medição de produtividade em times ágeis. Trabalhos futuros relacionados à aplicação da mesma abordagem em um time ou projeto que obtenha resultados satisfatórios pode ser válido para concretização da conclusão obtida neste trabalho. Além disso, a utilização dos processos em outros tipos de modelos de estimativa pode trazer um contexto generalista e dinâmico para a abordagem estabelecida. Além disso, a criação de um *software*, como ferramenta para utilizar o COCOMO II, com as práticas atuais pode ser válido para

implementação das fases e cálculos de calibração através de modelos de regressão ou outros mais avançados.

REFERÊNCIAS

ANDERSON, David J. **Kanban: successful evolutionary change for your technology business**. Sequim, Washington: Blue Hole Press, 2010.

BECK, Kent; ANDRES, Cynthia. **Extreme programming explained: embrace change**. 2nd ed. Boston, MA: Addison-Wesley, 2005.

BECK, Kent; BEEDLE, Mike; VAN BENNEKUM, Arie; *et al*. **Manifesto for Agile Software Development**. 2001.

BOEHM, Barry W. (Org.). **Software cost estimation with Cocomo II**. Upper Saddle River, NJ: Prentice Hall, 2000.

CAPRETZ, Luiz Fernando. Personality types in software engineering. **International Journal of Human-Computer Studies**, v. 58, n. 2, p. 207–214, 2003.

DE RORE, Lotte; SNOECK, Monique; POELS, Geert; *et al*. COCOMO II as Productivity Measurement: A Case Study at KBC. **SSRN Electronic Journal**, 2008.

DESTEFANIS, Giuseppe; ORTU, Marco; COUNSELL, Steve; *et al*. Software development: do good manners matter? **PeerJ Computer Science**, v. 2, p. e73, 2016.

DINGSØYR, Torgeir; LINDSJØRN, Yngve. Team Performance in Agile Development Teams: Findings from 18 Focus Groups. *In*: BAUMEISTER, Hubert; WEBER, Barbara (Orgs.). **Agile Processes in Software Engineering and Extreme Programming**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, v. 149, p. 46–60. (Lecture Notes in Business Information Processing).

FATEMA, Israt; SAKIB, Kazi. Factors Influencing Productivity of Agile Software Development Teamwork: A Qualitative System Dynamics Approach. *In*: **2017 24th Asia-Pacific Software Engineering Conference (APSEC)**. Nanjing: IEEE, 2017, p. 737–742.

FENTON, Norman E; BIEMAN, James. **Software metrics: a rigorous and practical approach**. [s.l.: s.n.], 2020.

FERNANDES, Jorge Henrique Cabral. Qual a prática do desenvolvimento de software? **Ciência e Cultura**, v. 55, p. 29–33, 2003.

JOHNSON, Jim. CHAOS 2020: Beyond Infinity. **Standish Group**, 2020.

JONES, Capers. **Programming productivity**. New York: McGraw-Hill, 1986. (McGraw-Hill series in software engineering and technology).

JULIA, Ana; FORNO, Ana. IMPLEMENTAÇÃO DE KANBAN DE FORNECEDOR, TRANSPORTE E PRODUÇÃO: ESTUDO DE CASO EM EMPRESA DE CABINES DE MÁQUINAS AGRÍCOLAS. 2022.

KIROVSKA, Nevenka; KOCESKI, Saso. Usage of Kanban methodology at software development teams. **Journal of applied economics and business**, v. 3, n. 3, p. 25–34, 2015.

KNIBERG, Henrik; SKARIN, Mattias. **Kanban and Scrum: making the most of both**. s. l.: C4Media, 2010.

LAUREANO, Raul M. S.; CAETANO, Nuno; CORTEZ, Paulo. Previsão de tempos de internamento num hospital português: aplicação da metodologia CRISP-DM. **Iberian Journal of Information Systems and Technologies**, v. 0, n. 13, p. 83–98, 2014.

LEAU, Yu Beng; LOO, Wooi Khong; THAM, Wai Yip; *et al*. Software development life cycle AGILE vs traditional approaches. *In*: **International Conference on Information and Network Technology**. [s.l.: s.n.], 2012, v. 37, p. 162–167.

LEFFINGWELL, Dean. **Scaling software agility: best practices for large enterprises**. Upper Saddle River, NJ: Addison-Wesley, 2007. (The Agile software development series).

NIKIFOROVA, Oksana; NIKULSINS, Vladimirs; SUKOVSKIS, Uldis. Integration of MDA framework into the model of traditional software development. *In*: **Databases and Information Systems V**. [s.l.]: IOS Press, 2009, p. 229–239.

PRESSMAN, Roger S. **Software engineering: a practitioner's approach**. Eighth edition. New York, NY: McGraw-Hill Education, 2015.

REEVES, Jack W. What is software design?. **C++ Journal**, v. 2, n. 2, p. 14–12, 1992.

SHARMA, Sheetal; SARKAR, Darothi; GUPTA, Divya. Agile processes and methodologies: A conceptual study. **International journal on computer science and Engineering**, v. 4, n. 5, p. 892, 2012.

SHEARER, Colin. The CRISP-DM model: the new blueprint for data mining. **Journal of data warehousing**, v. 5, n. 4, p. 13–22, 2000.

SILVA, Diogo; SANTOS, F.; NETO, Pedro Santos. Os benefícios do uso de Kanban na gerência de projetos de manutenção de software. *In*: **Anais do VIII Simpósio Brasileiro de Sistemas de Informação**. Porto Alegre, RS, Brasil: SBC, 2012, p. 715–725.

SOMMERVILLE, Ian. **Software engineering**. Tenth edition. Boston: Pearson, 2016.

STEYAERT, Patrick. **Essential Upstream Kanban.** s. l.: Lean-Kanban University, 2018.

SYMONS, C.R. Function point analysis: difficulties and improvements. **IEEE Transactions on Software Engineering**, v. 14, n. 1, p. 2–11, 1988.

VIJAYASARATHY, Leo R.; TURK, Dan. Agile software development: A survey of early adopters. **Journal of Information Technology Management**, v. 19, n. 2, p. 1–8, 2008.

WIRTH, Rüdiger. CRISP-DM: Towards a standard process model for data mining. *In*: **Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining**. [s.l.: s.n.], 2000, p. 29–39.