Estimativa de Esforço em Projetos Ágeis de Software Utilizando Mapas de Kohonen

Werney Ayala Luz Lira[1], Francisco Vanderson de Moura Alves[1], Pedro de Alcântara dos Santos Neto[1], Ricardo de Andrade Lira Rabêlo[1], Ricardo de Sousa Britto[1]

[1]EASII - Laboratório de Engenharia de Software e Informática Industrial
DC - Departamento de Computação
CCN - Centro de Ciências da Natureza
UFPI - Universidade Federal do Piauí
Brasil

werney.zero@gmail.com, vanderson.moura@yahoo.com.br

pasn@ufpi.edu.br, ricardoalr@ufpi.edu.br, rbritto@ufpi.edu.br

Abstract. *Software development using agile methods is directly associated to the execution of several activities planned in the iterations. A suitable time estimation makes the project stable, the development team safer and the customer more satisfied with the deliveries. However, estimating the time for development of a task is somewhat error prone, since all the techniques have some practical limitations of use. This paper aims to conduct a study on the use of Kohonen self-organizing maps to assist in these estimates, providing an initial estimate of the duration of a task, based on measurements obtained using similar tasks previously performed. To evaluate the approach we used data obtained from a local software development company. From the evaluation was possible to draw conclusions on the quality of records made by the developers of the company, and evaluate the accuracy of the estimate made by the approach.*

Resumo. *O desenvolvimento de software por meio de métodos ágeis está diretamente ligado à execução das várias atividades planejadas ao longo das iterações. Boas estimativas de prazos tornam o projeto estável, a equipe de desenvolvimento mais segura e o cliente mais satisfeito. No entanto, estimar o prazo de desenvolvimento de uma tarefa é algo sensível a erros, uma vez que todas as técnicas possuem alguma limitação prática de uso. Este trabalho tem como objetivo a realização de um estudo sobre o uso de mapas auto-organizáveis de Kohonen para auxiliar nessas estimativas, fornecendo uma estimativa inicial do tempo de duração de uma tarefa, utilizando como base medidas obtidas de tarefas semelhantes e previamente realizadas. Para avaliar a abordagem foram utilizados dados obtidos de uma empresa de desenvolvimento de software local. A partir da avaliação foi possível tirar conclusões relativas à qualidade dos registros feitos pelos desenvolvedores dessa empresa, e avaliar a precisão da estimativa feita pela abordagem.*

1. Introdução

A engenharia de software tem como principal objetivo auxiliar o desenvolvimento, a documentação e a manutenção de software, garantindo a qualidade do produto gerado.

Para isso existem vários processos de desenvolvimento que especificam detalhadamente cada fase de um projeto, descrevendo o que deve ser feito, como deve ser feito, quando deve ser feito e por quem deve ser feito [Pressman 2010]. Alguns processos apresentam especificações resumidas, na forma de diretrizes, sendo conhecidos como métodos ágeis [Beck et al. 2001]. Existem contextos em que tais abordagens são muito apropriadas, especialmente em casos no qual os prazos são curtos ou quando o nível de incerteza, por parte do cliente, é alto.

Os processos baseados no modelo incremental são divididos em ciclos regulares de trabalho ao longo do tempo, denominados iterações. As iterações duram em média de 2 a 4 semanas e têm como resultado uma versão incremental e potencialmente utilizável do produto. Antes do início de uma nova iteração é realizada uma reunião visando planejar as tarefas a serem desenvolvidas, utilizando como base para essa escolha a lista de requisitos existentes. Durante a reunião de planejamento os requisitos são discutidos e a partir disso, são geradas várias tarefas a serem executadas visando ao seu atendimento [Schwaber and Beedle 2001].

Após realizar a seleção de quais tarefas serão desenvolvidas em uma iteração, é necessário estimar a duração de cada uma delas e por fim atribui-las aos desenvolvedores disponíveis na equipe. No entanto, a estimativa de realização de uma tarefa depende, dentre vários fatores, até mesmo de quem vai realizá-la, fato esse que pode gerar um grande número de combinações de possíveis alocações tarefa/desenvolvedor até encontrar o cenário ideal, com a melhor alocação desenvolvedor/tarefa.

Uma estimação mal feita impacta diretamente nos prazos do projeto. Caso a estimativa seja abaixo do tempo real de duração da tarefa, os desenvolvedores podem ser sobrecarregados para cumprir os prazos acertados. Caso a estimativa seja acima do tempo real de duração da tarefa, poder-se-á gerar um custo excessivo ao projeto levando à uma duração maior do que a necessária para sua conclusão, o que causa mais gasto para a empresa desenvolvedora.

Visto que a estimação de uma tarefa é de grande importância para um projeto de software, erros de planejamento podem gerar graves consequências, tanto para o projeto, quanto para a empresa e até mesmo para os clientes. Nesse contexto, a existência de abordagens automatizadas que possam auxiliar a equipe de desenvolvimento de software nessa atividade mostra-se de grande valor. Mesmo sendo de fundamental importância para o projeto, a precisão das métricas utilizadas não ultrapassa o limiar de 25% de precisão, como foi relatado em [Usman et al. 2014].

Este trabalho se propõe a apoiar a estimação da duração de uma tarefa, de modo a servir como uma base para uma estimação mais precisa por parte da equipe de desenvolvimento. A ideia inicial foi iniciar um estudo nessa direção, culminando com a proposição de um método para incorporar tais técnicas no planejamento de tarefas de uma iteração. Essa abordagem utiliza uma rede neural artificial [Haykin 1998], mais precisamente um mapa auto-organizável de Kohonen [Kohonen 1982], de modo a fornecer o tempo estimado de duração da tarefa a partir do tempo gasto em tarefas anteriores com características semelhantes.

Como foi explicado anteriormente, o processo de estimação da duração de uma tarefa é uma atividade que está mais relacionada ao desenvolvimento ágil, sendo realizada

antes do inicio de uma nova iteração. Desse modo, a abordagem proposta neste trabalho demonstra ter um maior valor no contexto ágil. Porem, ela também pode ser aplicada em um projeto que utiliza desenvolvimento tradicional, desde que seja possível extrair do projeto as informações necessárias para a execução da abordagem.

Este trabalho está organizado da seguinte forma: a Seção 2 apresenta alguns trabalhos relacionados; a Seção 3 apresenta o referencial teórico, que descreve alguns conceitos necessários para um bom entendimento do trabalho; a Seção 4 descreve a abordagem proposta; a Seção 5 apresenta uma aplicação da abordagem em um contexto real e os resultados obtidos; por último, a Seção 6 apresenta a conclusão e direções para trabalhos futuros.

2. Trabalhos Relacionados

Existem várias técnicas para estimar esforço em software, dentre elas podemos destacar COCOMO (*Constructive Cost Model*) [Boehm 1981], SLIM (*Software Life Cycle Management*) [Putnam 1978], além da utilização de técnicas de inteligência computacional como, redes neurais artificiais, lógica *fuzzy*, árvores de decisão, dentre outras. Em [Rastogi et al. 2014] é apresentado um Survey contendo as principais técnicas e modelos utilizados para realizar estimativas de esforço no desenvolvimento de software no últimos anos.

O autores em [Kumar et al. 2011] apresentaram um método para estimativa de tempo (esforço) requerido para a construção de software utilizando lógica *fuzzy*. O modelo proposto tem como entrada o tamanho do software e como saída o esforço necessário para sua construção. O tamanho é medido em KLOC e o esforço é estimado em homens/mês. Foi escolhido lógica *fuzzy* por sua característica de lidar com imprecisão e obscuridade dos dados. O método proposto foi avaliado e comparado com vários outros como, COCOMO, *early design*, *post Arch*, dentre outros, e mostrou-se bastante efetivo, obtendo resultados melhores em comparação com esses outros métodos. Para a avaliação foram utilizados dados obtidos de [Kemerer 1987].

Em [Satapathy et al. 2013] é apresentado um modelo para estimativa do esforço requerido para desenvolver vários projetos de software utilizando uma abordagem de pontos de classe. A otimização dos parâmetros de esforço é conseguido a partir de uma regressão adaptativa baseada em uma rede neural artificial de múltiplas camadas (*Multilayer Perceptron* - MLP). A avaliação foi feita utilizando dados de 40 projetos de estudantes, desenvolvidos em java. Os resultados obtidos foram comparados com os obtidos a partir da utilização de um outro modelo de rede neural, a rede de base radial (*Radial Basis Function* - RBF). Concluiu-se que a rede MLP obtém uma precisão melhor em suas estimativas.

Os autores em [Attarzadeh and Ow 2011] propuseram um método para melhorar a precisão da estimativa de esforço do COCOMO II, a partir do uso de lógica *fuzzy*. Segundo eles a lógica *fuzzy* ajuda a reduzir a incerteza e a imprecisão de alguns atributos. Sistemas *fuzzy* são utilizados nas entradas do sistema de estimativa. Segundo os autores esse processo ajuda a reduzir a imprecisão e incerteza em relação aos dados. Após a etapa anteriormente descrita, é calculado o esforço utilizando o COCOMO II, que retorna o valor estimado para o custo, tempo e quantidade de pessoas/mês. Para avaliar o método são utilizados dados obtidos do COCOMO I, da NASA e de duas empresas de software.

Os resultados foram comparados com a utilização apenas do COCOMO II e percebeu-se uma melhora a partir da aplicação de lógica *fuzzy* no modelo.

Os autores em [Saxena and Singh 2012] apresentaram um método de estimativa de esforço utilizando uma abordagem neuro-fuzzy, ou seja, uma método híbrido que combina a capacidade de aproximação de funções da rede neural com um sistema *fuzzy* para aumentar a precisão na aproximação de funções não lineares. O método proposto foi comparado com outras técnicas clássicas utilizadas para estimativa de esforço em projetos de software, tais como, *Haslted*, *Walston-Felix*, e outros. Para isso foram utilizados dados obtidos da NASA. A partir da avaliação percebeu-se que o método proposto é bastante eficaz, e ligeiramente melhor que o modelo de *Bailey-Basili*.

Em [Manapian and Prompoon 2014] é proposto um modelo de estimativa de tempo de desenvolvimento de software para as mudanças ocorridas nos requisitos durante a fase de implementação, baseado na análise de perfis de protótipo, a fim de definir os fatores que podem afetar o tempo de desenvolvimento de software. Esse modelo foi desenvolvido utilizando o *Analogy Estimation Method*, que utiliza dados históricos de projetos similares anteriores. A metodologia proposta foi dividida em duas partes, na primeira é feito a análise dos fatores e na segunda é proposto um modelo de estimativa de tempo utilizando o *Analogy Estimation Method*. O modelo proposto não foi comparado com outros métodos, mas chegaram à conclusão de que o modelo é aplicável a partir dos valores de PRED (*Percentage Relative Error Deviation*) e MMRE (*Mean Magnitude Relative Error*), que são as duas medidas de precisão mais utilizadas pela literatura [Conte et al. 1986].

Em [Usman et al. 2014] foi realizado uma revisão sistemática do que foi feito relacionado à estimativa de esforço no contexto de métodos ágeis. A partir da revisão sistemática percebeu-se que são utilizadas várias métricas para medir a precisão da estimativa realizada, tais como MMRE, o índice Kappa foi utilizado para medir a concordância entre as medidas, mas na maioria dos casos a precisão não atinge o limiar de 25%. Percebeu-se também que a maioria das técnicas para estimar esforço utilizadas são baseadas em alguma forma de avaliação subjetiva baseada em especialistas. Como será abordado nas seções posteriores, os resultados obtidos neste trabalho também não foram muito precisos, por outro lado a abordagem aqui proposta é quase totalmente automatizada. Os especialistas são necessários apenas para definição das classes.

Outro ponto importante é que a maioria dos métodos propostos se concentram na estimativa de custos. A estimativa de tempo portanto carece de mais métodos e alternativas para sua realização. Neste trabalho é proposto o uso de mapas auto-organizáveis de Kohonen para auxiliar no processo de estimação da duração de tarefas, consequentemente diminuindo os erros durante essa fase, diminuindo também eventuais prejuízos provocados por atrasos no projeto.

3. Referencial Teórico

3.1. Métricas de Software

Para medir a qualidade ou o esforço para construção de um software e com isso inferir o custo do mesmo geralmente são utilizadas métricas. Existem várias métricas para medir aspectos ou características de um software. Neste trabalho serão abordadas apenas as

métricas de código-fonte, ou seja, as métricas que tratam características do código-fonte de um software, mais precisamente as métricas de tamanho e complexidade.

As métricas de tamanho tentam quantificar o tamanho de um software. As mais utilizadas são: linhas de código (LOC) [Boehm 1981, Jones 1986, Jones 1991] e pontos de função (FP) [Albrecht and Gaffney 1983].

As métricas de complexidade são utilizadas para medir a complexidade de um trecho de software. Trecho com uma alta complexidade devem ser modularizados de modo a diminuir essa complexidade. Geralmente trechos grandes tendem a ser mais complexos, por isso o uso conjunto dessas duas métricas, de tamanho e de complexidade, é bastante recomendado pela comunidade. As métricas de complexidade mais utilizadas são: complexidade ciclomática ($v(G)$) [McCabe 1976], número de nós e fluxo de informações.

Neste trabalho serão utilizadas apenas uma métrica de tamanho e uma de complexidade. As métricas escolhidas foram: tamanho em linhas de código e complexidade ciclomática. Essas métricas foram escolhidas por serem simples e estarem mais próximas da realidade da maioria das empresas de desenvolvimento de software. No entanto, nada impede que qualquer uma das outras métricas sejam utilizadas, se assim for desejado.

LOC provavelmente é a métrica mais simples utilizada para medir o tamanho de um software. A contagem é feita apenas em linhas que contenham verdadeiramente código, ou seja, linhas em branco e comentário não são contabilizados.

A complexidade ciclomática mede a complexidade de um trecho de código por meio da contabilização do número de caminhos que aquele trecho de código possui. O número de caminhos está diretamente relacionado ao número de estruturas de controle presente no código. Por exemplo, um trecho de código sem nenhuma estrutura de controle tem complexidade ciclomática um. Caso seja adicionado um *if* a esse trecho de código a complexidade passará a ser dois, pois existirão dois caminhos, um se a condição contida no *if* for satisfeita e outro caso contrário.

3.2. Mapas Auto-Organizáveis de Kohonen

Os mapas auto-organizáveis foram propostos por Teuvo Kohonen na década de 80 [Kohonen 1982]. Um mapa auto-organizável é um modelo de rede neural artificial, bio-inspirado no funcionamento do córtex cerebral, que é capaz de diferenciar os diversos estímulos produzidos pelo corpo, tais como, estímulos sensoriais, motores ou auditivos e, direcioná-los para suas áreas correspondentes no cérebro. O treinamento da rede é feito de modo não supervisionado, ou seja, quando não há a informação do valor de saída desejada nas amostras de treinamento.

Os neurônios são organizados em uma grade normalmente uni ou bi-dimensional. Nessa grade os neurônios competem entre si para definir qual neurônio, ou conjunto de neurônios, serão ativados. O neurônio que vencer essa competição é chamado de "neurônio vencedor" e como prêmio por essa vitória esse neurônio terá os seus pesos sinápticos ajustados. Podem ser adotadas duas estratégias de ajuste dos pesos, a primeira é conhecida como "o vencedor leva tudo". Nessa estratégia apenas os pesos sinápticos do neurônios vencedor são ajustados. Uma segunda estratégia seria ajustar os pesos sinápticos dos vizinhos do neurônio vencedor proporcionalmente à distância entre o vizinho e o vencedor, dessa forma neurônios próximos tendem a responder estímulos seme-

lhantes.

O processo de treinamento se divide em três etapas: competição, cooperação e adaptação. Na etapa de competição os neurônios da camada de saída são estimulados afim de definir o neurônio vencedor. Na etapa de cooperação é definida a localização espacial de uma vizinhança topológica. Na etapa de adaptação os pesos sinápticos do neurônios vencedor e dos seus vizinhos são ajustados com o objetivo de melhorar a resposta desses neurônios frente a aplicação subsequente de um padrão de entrada similar [Haykin 1998].

As três etapas do processo de treinamento são executadas repetidas vezes para cada amostra do conjunto de treinamento. A atividade de executar o treinamento para todas as amostras do conjunto de treinamento recebe o nome de época. Para considerar uma rede neural treinada são necessárias várias épocas, ou seja, o conjunto de treinamento é utilizado várias vezes durante esse processo.

Os mapas auto-organizáveis são utilizados em problemas de clusterização (agrupamento), pois o principal objetivo dessa técnica é agrupar os dados de entrada a partir de alguma similaridade entre eles, formando classes ou grupos, ou ainda *clusters*. É importante enfatizar que nem sempre uma classe será representada por um único neurônio do mapa. Devido a influência da vizinhança no processo de treinamento pode ser que um conjunto de neurônios próximos reconheçam o mesmo padrão, definindo assim uma mesma classe.

4. Abordagem Proposta

A abordagem proposta neste trabalho auxilia na estimação da duração de tarefas em projetos de desenvolvimento de software. Ela está dividida em quatro etapas, como pode ser visto na Figura 1.

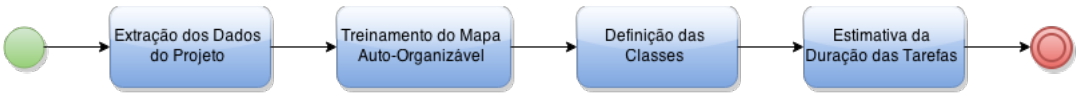

Figura 1. Abordagem Proposta

A primeira etapa refere-se à extração de dados de um projeto real, visando com isso realizar uma clusterização das tarefas executadas, para servir de base para a segunda etapa, que é o treinamento da rede. A terceira etapa consiste na definição das classes a servir de apoio para as estimativas. Finalmente, a quarta etapa é estimar a duração das tarefas, com base na rede previamente treinada e nas classes que foram definidas. Em suma, as etapas da abordagem são:

- Etapa 1: Extração dos dados do projeto.
- Etapa 2: Treinamento da rede neural.
- Etapa 3: Definição das Classes
- Etapa 4: Estimativa da duração das tarefas.

As Etapas 1, 2 e 3 são executadas com dados oriundos de tarefas realizadas anteriormente, pois visam o treinamento do mapa auto-organizável. Para isso utiliza-se informações precisas (valores numéricos de complexidade e tamanho) extraídos do projeto. Já a Etapa 4 é executada com informações das novas tarefas. Para isso utiliza-se

informações não tão precisas (valores qualitativos informados pelos desenvolvedores). Esses valores qualitativos podem ser por exemplo: complexidade alta, tamanho pequeno, etc., que são baseados nas classes definidas na Etapa 3.

4.1. Etapa 1: Extração dos Dados do Projeto

A extração de dados do projeto é a base para o treinamento de uma rede neural, mais precisamente um mapa auto-organizável de Kohonen, que será utilizado para agrupar tarefas semelhantes. Espera-se com isso que os grupos de tarefas semelhantes tenham também duração semelhante, auxiliando no processo de estimação. Essa extração é feita a partir de projetos executados em uma empresa e possui como restrição a existência de tarefas planejadas, com o tempo gasto durante a sua realização, além da ligação dessa tarefa com o repositório de código utilizado na organização, para que seja possível coletar dados de tamanho e complexidade das tarefas.

Conforme já comentado, duas métricas são necessárias para a execução da abordagem aqui proposta: uma métrica de complexidade (foi escolhido a complexidade ciclomática) e uma métrica de tamanho (foi escolhido tamanho em linhas de código).

A complexidade ciclomática é contabilizada para cada arquivo contido no projeto e associado a uma tarefa executada. Isso acontece por que, ao realizar uma tarefa, o desenvolvedor deve entender o código pré-existente afim de fazer suas intervenções no código, ou seja, ele deve compreender os métodos/funções contidas no módulo a ser atacado e isso está diretamente ligado ao tempo para realizar tais intervenções. Já o tamanho em linhas de código refere-se ao trabalho efetivo do desenvolvedor, ou seja, a quantidade de linhas que ele alterou ou adicionou ao código, a partir da execução de uma tarefa previamente planejada. Isso é calculado por meio de uma análise do sistema de controle de versão da empresa. São analisados os *commits* feitos pelos desenvolvedores e que estão associados a uma tarefa, para que seja contabilizado exclusivamente a quantidade de alterações no código proveniente da tarefa executada.

Extrair essas métricas manualmente pode levar muito tempo. Para isso foi criado uma ferramenta para extração automática da complexidade ciclomática e das linhas de código associadas às tarefas planejadas em uma iteração. A necessidade dessa ligação entre a tarefa planejada e o repositório de código é fundamental para esta abordagem, uma vez que isso indicaria o tamanho e a complexidade granular gerada por cada tarefa. No entanto, conforme já frisado anteriormente, apenas uma empresa dentre as dezenas de empresas consultadas possui esse nível de organização, fato esse que tornou a extração de dados limitado a um único contexto.

4.2. Etapa 2: Treinamento da Rede Neural

De posse das informações de complexidade ciclomática e tamanho em linhas de código das tarefas realizadas anteriormente é feito então o treinamento da rede neural, que por sua vez realiza a clusterização das amostras, separando-as em grupos. A finalidade desta etapa é agrupar tarefas semelhantes, levando em consideração alguns atributos básicos extraídos na etapa anterior: tempo real gasto na tarefa, complexidade e tamanho da porção de código associada à tarefa.

Os grupos gerados representam um conhecimento para a empresa, uma vez que exibem elementos com comportamentos similares. Isso pode ser bastante explorado para

se encontrar gargalos no desenvolvimento de software (tarefas com nível de erro muito alto), além de permitir a inferência de tipos de tarefas que possuem um bom nível de acerto de estimativa. Essas informações também são base para a criação de um bom mecanismo de estimação, que é o objetivo final desta pesquisa.

4.3. Etapa 3: Definição das Classes

O processo de definição das classes classes nada mais é do que atribuir um rótulo à cada grupo identificado na etapa anterior. É importante ressaltar que nem sempre um neurônio isolado representa um grupo, pode haver casos de mais de um neurônio representarem um mesmo grupo. Isso acontece quando as amostras identificadas por esses neurônios são muito parecidas.

Neste trabalho os rótulos foram criados baseados na própria interpretação do conceito que o grupo possui. Por exemplo, grupos que identificam amostras que possuem complexidade baixa e tamanho pequeno receberam esse identificador ("Complexidade Baixa e Tamanho Pequeno") como rótulo. Os valores que definem que uma amostra tem complexidade baixa e tamanho pequeno são definidos durante a aplicação da abordagem, pois esses valores podem variar de acordo com o problema a ser resolvido.

4.4. Etapa 4: Estimativa da Duração das Tarefas

Tendo como base uma organização em grupos das tarefas existentes, é possível estabelecer heurísticas para se estimar tempo de duração de novas tarefas. Mas para isso é necessário que existam informações adicionais, mesmo que imprecisas (qualitativas), com relação às tarefas que se deseja estimar. Informações tais como complexidade alta e tamanho pequeno são úteis nessa etapa, uma vez que isso tende a auxiliar a descoberta do grupo mais associado à tarefa a ser executada. Uma vez estabelecida essa similaridade, é possível usar como base a média das durações das tarefas desse grupo. Esse valor calculado pode ajudar na estimação de tempo para realização da nova tarefa.

5. Aplicação da Abordagem e Resultados

5.1. Objetivos

A aplicação da abordagem em um contexto real tem como objetivos:

- Objetivo 1: investigar a qualidade dos registros de durações das tarefas mantidos por essa empresa.
- Objetivo 2: investigar a distribuição dessas tarefas em relação à complexidade e tamanho.
- Objetivo 3: compreender a relação entre o aumento nos valores de complexidade e tamanho com o aumento no tempo de duração das tarefas.

5.2. Extração dos Dados do Projeto

Infelizmente, a obtenção dos dados necessários para a execução da abordagem foi bastante complicada. A equipe responsável por este projeto contactou fábricas de software em diferentes locais do Brasil, além de ter adquirido duas bases de dados históricos de projetos internacionais, mas nenhuma atendia às restrições para uso neste projeto. Apenas uma empresa dentre as dezenas de empresas contatadas atendeu às restrições (tarefas planejadas ligadas ao repositório de código). Esse fato foi um grande limitador da pesquisa

e revela como as empresas ainda possuem dificuldades na implementação do conceito de rastreabilidade em projetos de desenvolvimento.

Durante esta avaliação foram utilizadas informações de 140 tarefas dessa empresa. Deve-se ressaltar a dificuldade de obter dados para a avaliação, pois são necessários informações relativas à complexidade ciclomática e tamanho em linhas de código de cada tarefa, além do tempo gasto e planejado para realizá-la. As duas primeiras informações são fáceis de serem obtidas dos *commits* nos repositórios de código, mas nem sempre é possível saber qual a tarefa associada, uma vez que nela é que existe a informação do tempo efetivamente gasto e do planejado, além do problema de que dois *commits* diferentes podem se referir a uma mesma tarefa.

A empresa em questão mantinha um certo rigor sobre a rastreabilidade de suas tarefas. Os *commits* no repositório de código estavam ligados às tarefas registradas em uma ferramenta web própria para esta finalidade (Redmine[1]). Desse modo era possível definir a qual tarefa um determinado *commit* pertencia e o próprio Redmine mantinha um registro do tempo gasto para a realização de cada tarefa.

Para realizar esta avaliação era necessário que os dados utilizados estivessem completos, ou seja, não poderia faltar nenhuma informação relativa à complexidade ciclomática, quantidade de linhas de código alteradas ou tempo gasto durante a realização de cada tarefa. Como a extração dessas informações foi feita de forma automática, existia a possibilidade de alguns desses valores ser zero. As tarefas que continham valores nulos em algum dos seus atributos foram removidas.

5.3. Treinamento do Mapa Auto-organizável

Os valores de entrada foram normalizados em uma faixa de valores entre zero e um. Para maior eficácia do método foram aplicadas, durante a fase de ordenação, uma taxa de aprendizado alta e uma vizinhança grande, que foram diminuídas gradativamente, ao longo das 1000 épocas dessa fase. Isso é feito afim de promover uma aproximação mais rápida dos neurônios semelhantes. A taxa de aprendizado diminui seguindo a Equação 1.

$$\eta(n) = \eta_0 * exp\left(\frac{-n * \log(100 * \eta_0)}{n_{ord}}\right) \quad (1)$$

Onde:

- n - época atual;
- $\eta(n)$ - valor da taxa de aprendizado para a época n;
- η_0 - valor da taxa de aprendizado definido durante a inicialização do algoritmo;
- n_{ord} - quantidade total de épocas da fase de ordenação;

Já o raio da vizinhança diminui seguindo a Equação 2.

$$r(n) = r_0 * exp\left(\frac{n * \log(r_0)}{n_{ord}}\right) \quad (2)$$

Onde:

[1]www.redmine.org

- n - época atual;
- $r(n)$ - valor do raio para a época n;
- r_0 - valor do raio definido durante a inicialização do algoritmo;
- n_{ord} - quantidade total de épocas da fase de ordenação;

A taxa de aprendizado foi inicializada com o valor de 0,1 e diminuindo gradativamente, mas se mantendo acima de 0,01. O raio da vizinhança foi inicializado de modo a incluir quase todos os neurônios do mapa, mantendo-se acima de 1 durante a fase de ordenação.

A fase de convergência teve ao todo 500 épocas. Durante essa fase tanto a taxa de aprendizado quanto a vizinhança mantiveram-se em um valor fixo. A taxa de aprendizado manteve-se em 0,01 e a vizinhança foi definida como sendo nula (menor que 1), incluindo apenas o neurônio vencedor. Foram feitos vários testes com diferentes valores desses parâmetros e os melhores resultados foram obtidos a partir do uso desses valores.

O mapa auto-organizável contendo 16 neurônios dispostos em uma grade bidimensional pode ser visto na Figura 2. Foi definido essa quantidade de neurônios devido a quantidade de amostras (tarefas), desse modo cada neurônio identificaria cerca nove amostras. Uma quantidade maior de neurônios poderia acarretar em grupos muitos pequenos, com até mesmo uma única amostra.

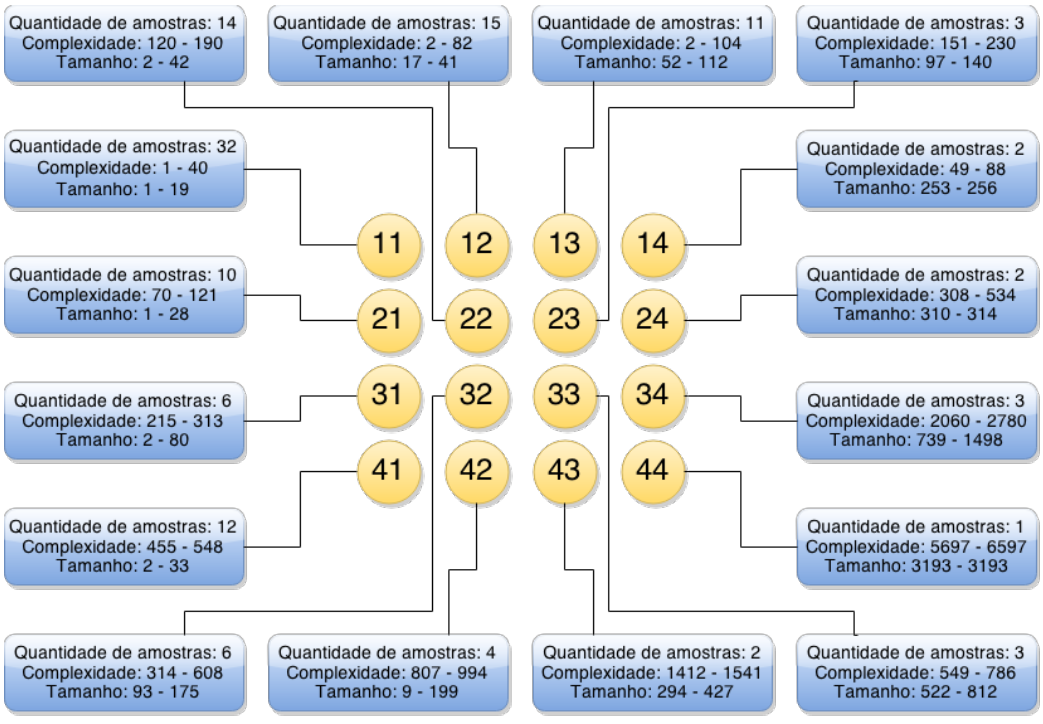

Figura 2. Disposição dos neurônios do mapa auto-organizável

Após o treinamento do mapa auto-organizável percebe-se que grande parte das amostras (tarefas) possuem tamanho pequeno (variando entre 1 e 19 linhas alteradas) e complexidade baixa (variando entre 1 e 40). Essas amostras são reconhecidas pelo neurônio "11" do mapa. Como pode-se perceber, as tarefas menos complexas e menores

são reconhecidas pelos neurônios do canto superior esquerdo do mapa e as tarefas mais complexas e maiores são reconhecidas pelos neurônios do canto inferior direito do mapa.

5.4. Definição das Classes

Após o treinamento é necessário definir qual neurônio, ou conjunto de neurônios, representam uma determinada classe. Para esta avaliação foram definidas 9 classes, que representam todas as combinações possíveis entre complexidade baixa, média ou alta e tamanho pequeno, médio ou grande. Desse modo uma possível classe seria "complexidade baixa e tamanho pequeno" e assim por diante até completar todas as combinações. A Tabela 1 exibe que classe cada neurônio representa.

Neurônio	Classe
11	Complexidade Baixa e Tamanho Pequeno
12	Complexidade Baixa e Tamanho Pequeno
13	Complexidade Baixa e Tamanho Pequeno
14	Complexidade Baixa e Tamanho Médio
21	Complexidade Média e Tamanho Pequeno
22	Complexidade Média e Tamanho Pequeno
23	Complexidade Média e Tamanho Médio
24	Complexidade Média e Tamanho Médio
31	Complexidade Média e Tamanho Pequeno
32	Complexidade Média e Tamanho Médio
33	Complexidade Média e Tamanho Médio
34	Complexidade Alta e Tamanho Grande
41	Complexidade Média e Tamanho Pequeno
42	Complexidade Alta e Tamanho Pequeno
43	Complexidade Alta e Tamanho Médio
44	Complexidade Alta e Tamanho Grande

Tabela 1. Definição das classes

Nota-se que nem todas as classes possuem neurônios associados, como "complexidade baixa e tamanho grande" e "complexidade média e tamanho grande". Isso é particularidade das tarefas dessa empresa, assim como o fato da maioria das tarefas serem de complexidade baixa e tamanho pequeno.

As classes foram definidas seguindo a seguinte definição feita pelo gerente de projeto após analisar os resultados. Cada neurônio reconhece amostras pertencentes à uma faixa de valores de complexidade e tamanho. Neurônios que reconhecem amostras com complexidade variando entre 1 e 100 identificam amostras com complexidade baixa. Caso essa faixa de valores esteja entre 101 e 600 identificam amostras com complexidade média e caso os valores sejam maiores que 600 identificam amostras com complexidade alta. Essa definição vale apenas para a complexidade. Para o atributo tamanho as faixas com valores variando de 1 a 100, 101 a 400 e maiores que 400 identificam amostras com tamanho pequeno, médio e grande respectivamente.

Após as definição das classes é possível perceber que mesmo com o aumento da complexidade e do tamanho das tarefas, o tempo gasto para realiza-las não é alterado

consideravelmente. As tarefas de complexidade baixa e tamanho pequeno identificadas pelo neurônio "11" têm durações próximas das tarefas de complexidade alta e tamanho grande, identificadas pelos neurônios "34" e "44", como mostra a Tabela 2. Outro ponto importante é que em um neurônio que identifica apenas tarefas com complexidade baixa, variando entre 1 e 40, e tamanho pequeno, variando entre 1 e 19, uma de suas tarefas durou mais de 26 horas, comprovando a instabilidade dos dados.

Neurônio 11				Neurônio 34	Neurônio 44
2 h 30 min	2 h 50 min	1 h 39 min	2 h 45 min	2 h 49 min	6 h 6 min
13 h 15 min	2 h 34 min	39min	1 h	1 h 39 min	
3 h 35 min	2 h 33 min	2 h 52 min	1 h 46 min	3 h 50 min	
36 min	1 h 16 min	4 h 6 min	2 h 18 min		
4 h 18 min	1 h 31 min	26 h 43 min	1 h 54 min		
2 h 1 min	5 h 32 min	57 min	3 h 17 min		
1 h 4 min	3 h 44 min	25 min	48 min		
2 h 54 min	1 h 18 min	2 h 58 min	4 h 49 min		

Tabela 2. Duração das tarefas identificadas pelos neurônios "11", "34" e "44"

Como o cálculo da duração de uma nova tarefa depende diretamente das durações das tarefas anteriores e como a duração não aumenta com o aumento da complexidade e do tamanho, isso implica que tarefas de complexidade alta e tamanho grande e podem ter estimativas bem próximas de tarefas de complexidade baixa e tamanho pequeno. Caso um tarefa realmente complexa e grande precise ser estimada, essa estimativa não será boa, pois foi influenciada pela duração das outras tarefas de complexidade alta e tamanho grande, que tem durações pequenas.

5.5. Estimativa da Duração das Tarefas

O próximo passo da execução da abordagem é realizar as estimativas de duração baseadas nas médias de cada grupo. Para isso são necessárias as informações de complexidade e tamanho de cada tarefa a ser estimada. A grande vantagem da abordagem é que não há a necessidade de uma informação precisa de complexidade e tamanho. Por exemplo, o desenvolvedor não terá que informar que a complexidade de uma tarefa é 59 e que terá de alterar 132 linhas de código nessa mesma tarefa, bastará apenas que ele informe que a tarefa tem complexidade baixa e tamanho médio. De posse dessas informações será identificado a qual grupo a tarefa pertence e estimada a sua duração.

Para avaliar a eficácia da abordagem foi utilizado um conjunto de tarefas previamente executadas. Os resultados podem ser vistos na Tabela 3. As colunas "Complexidade" e "Tamanho" mostram os valores reais de complexidade e tamanho da tarefa, a coluna "Classe" exibe a classe a qual a tarefa pertence, a coluna "Tempo Real" exibe a duração real da tarefa e a coluna "Tempo Estimado" exibe o tempo estimado para a tarefa pela abordagem.

Pode-se observar que os resultados não foram satisfatórios. Uma provável explicação para esse fato pode estar na qualidade dos registros feitos pelos desenvolvedores dessa empresa, que podem não refletir de forma adequada o tempo gasto nas tarefas. Isso acontece por inúmeros motivos, como por exemplo, esquecimento do cronômetro

Complexidade	Tamanho	Classe	Tempo Real	Tempo Estimado
35	41	Complexidade Baixa e Tamanho Pequeno	2 h 51 min	3 h 36 min
170	34	Complexidade Média e Tamanho Pequeno	7 h 7 min	12 h 52 min
1332	657	Complexidade Alta e Tamanho Médio	21 h 10 min	8 h 15 min
1414	4622	Complexidade Alta e Tamanho Grande	14 h 31 min	3 h 37 min

Tabela 3. Duração estimada para as tarefas

ligado durante a execução de uma tarefa, mesmo quando não se está trabalhando nela, fato esse que pode gerar registros com uma maior duração do que a real. Isso pode ser observado na Tabela 2. O neurônio "11" identificou uma tarefa de complexidade baixa e tamanho pequeno que durou cerca de 26 horas para ser concluída. Provavelmente, o desenvolvedor esqueceu o cronômetro ligado durante a realização dessa tarefa.

Outros atributos como quantidade de arquivos alterados e o desenvolvedor responsável pela tarefa foram utilizados na tentativa de melhorar os resultados obtidos, ou chegar a mais conclusões relacionadas à qualidade dos resultados obtidos. A partir desses atributos chegou-se a conclusão de que os novos atributos não foram muito úteis, pois a quantidade de tarefas por desenvolvedor não era suficientemente grande para realizar uma clusterização separadamente, já que é visível que cada desenvolvedor representa um *cluster*. A quantidade de arquivos alterados tem uma grande correlação com a complexidade ciclomática já que ela é calculada por arquivo, sendo portanto um atributo dispensável no processo de clusterização.

6. Conclusão e Trabalhos Futuros

Este trabalho apresentou uma abordagem para auxiliar equipes de desenvolvimento a estimar a duração de suas tarefas ao longo de um projeto de desenvolvimento de software. As principais contribuições deste trabalho são:

- Um método para estimar a duração das tarefas a partir da duração de tarefas semelhantes, utilizando informações pouco precisas;
- Um método para agrupar tarefas semelhantes, possibilitando a extração de informações sobre essas tarefas;

Uma das grandes vantagens da abordagem é a possibilidade de estimar o tempo de uma tarefa utilizando informações pouco precisas, como complexidade baixa e tamanho pequeno. Isso permite que a equipe tenha menos trabalho durante essa atividade, pois informar que a complexidade de uma tarefa é baixa e o seu tamanho é pequeno é bem mais fácil do especular uma valor exato de complexidade e tamanho.

A partir do agrupamento de tarefas semelhantes é possível perceber certos padrões nos dados. Por exemplo, nos dados utilizados neste trabalho percebemos a partir do agrupamento das tarefas que, mesmo com o crescimento da complexidade e do tamanho a duração dessas tarefas não era alterada significativamente.

Foi realizado uma prova de conceito em uma empresa de desenvolvimento de software. Embora os resultados de estimativas com o apoio da abordagem não tenham sido bastante assertivos, foi possível inferir que os registros de tempo da empresa apresentam grandes inconsistências, fato esse facilmente perceptível pela clusterização realizada. Isso pode ser facilmente utilizado para se adequar as estimativas realizadas pela equipe.

É importante ressaltar que os resultados obtidos também foram fortemente influenciados pela dificuldade em se encontrar dados para serem usados como base para a pesquisa. Foram contatadas diversas empresas, mas apenas uma mantinha uma rastreabilidade entre tarefas e código, de forma a tornar possível a obtenção do tempo real gasto e trechos de código efetivamente associados à tarefa executada.

Muito ainda pode ser feito nesta pesquisa. Uma direção é usar mais atributos para a classificação das tarefas, tornando os agrupamentos mais pormenorizados. Outro passo é melhorar o cálculo da duração das tarefas, utilizando também mais informações sobre a tarefa a ser estimada. No entanto, nada disso será possível se não obtivermos mais dados históricos para servir de base para a realização do trabalho.

7. Agradecimentos

Agradecemos à empresa Infoway Tecnologia e Gestão em Saúde LTDA pela importante contribuição para a realização desta pesquisa. Agradecemos também à CAPES pela bolsa de mestrado.

Referências

Albrecht, A. and Gaffney, J. E. (1983). Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering*, SE-9(6):639–648.

Attarzadeh, I. and Ow, S. H. (2011). Improving estimation accuracy of the cocomo ii using an adaptive fuzzy logic model. In *2011 IEEE International Conference on Fuzzy Systems (FUZZ)*, pages 2458–2464.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for agile software development.

Boehm, B. W. (1981). *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Conte, S. D., Dunsmore, H. E., and Shen, V. Y. (1986). *Software Engineering Metrics and Models*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition.

Jones, C. (1986). *Programming Productivity*. McGraw-Hill Series in Software Engineering & Technology. McGraw-Hill.

Jones, C. (1991). *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill, Inc., New York, NY, USA.

Kemerer, C. F. (1987). An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69.

Kumar, J., Rao, T., Babu, Y., Chaitanya, S., and Subrahmanyam, K. (2011). A novel method for software effort estimation using inverse regression as firing interval in fuzzy logic. In *2011 3rd International Conference on Electronics Computer Technology (ICECT)*, volume 4, pages 177–182.

Manapian, A. and Prompoon, N. (2014). Software time estimation model for requirements change based on software prototype profiles using an analogy estimation method. In *Computer Science and Engineering Conference (ICSEC), 2014 International*, pages 366–371.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320.

Pressman, R. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, USA, 7 edition.

Putnam, L. (1978). A general empirical solution to the macro software sizing and estimating problem. *Software Engineering, IEEE Transactions on*, SE-4(4):345–361.

Rastogi, H., Dhankhar, S., and Kakkar, M. (2014). A survey on software effort estimation techniques. In *2014 5th International Conference Confluence The Next Generation Information Technology Summit (Confluence)*, pages 826–830.

Satapathy, S., Kumar, M., and Rath, S. (2013). Class point approach for software effort estimation using soft computing techniques. In *2013 International Conference onAdvances in Computing, Communications and Informatics (ICACCI)*, pages 178–183.

Saxena, U. and Singh, S. (2012). Software effort estimation using neuro-fuzzy approach. In *2012 CSI Sixth International Conference on Software Engineering (CONSEG)*, pages 1–6.

Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Usman, M., Mendes, E., Weidt, F., and Britto, R. (2014). Effort estimation in agile software development: A systematic literature review. In *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*, PROMISE '14, pages 82–91, New York, NY, USA. ACM.