CENTRO UNIVERSITÁRIO FEEVALE

TIAGO OTÓ BAUER

PRODUTIVIDADE NO DESENVOLVIMENTO DE SOFTWARE

Novo Hamburgo
2009

TIAGO OTÓ BAUER

PRODUTIVIDADE NO DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de curso
apresentado como requisito parcial
à obtenção do grau de Bacharel
em Sistema de Informações pelo
Centro Universitário Feevale

Orientador: Eduardo Pretz

Novo Hamburgo
2009

TIAGO OTÓ BAUER

Trabalho de Conclusão do Curso de Sistemas de Informações, com título Produtividade no desenvolvimento de software, submetido ao corpo docente do Centro Universitário Feevale, como requisito necessário para obtenção do Grau Bacharel em Sistemas de Informações.

Aprovado por:

_____
Professor Orientador

_____
Professor (Banca examinadora)

_____
Professor (Banca examinadora)

Novo Hamburgo, 19 de dezembro de 2009

AGRADECIMENTOS

Agradeço...

...aos meus pais, Heitor e Adelaide, por todos os ensinamentos que me deram até aqui, por terem me apoiado durante todos estes anos ao meu lado desde o meu nascimento até hoje.

...ao meu irmão e grande amigo Rodrigo, que sempre me ajudo nos momentos em que mais precisei.

...a minha namorada Maiara, que me incentivou a concluir este curso, por sua compreensão nos momentos em que tive dúvidas, por estar ao meu lado em todos os momentos, por ser minha eterna amiga.

...aos meus avos, que foram muito importantes na formação dos meus pais e na minha.

...ao meu orientador Pretz, que sem ele não teria conseguido concluir este TCC.

...a toda minha família e todos os meus amigos e colegas do CSI da Feevale.

...a instituição de ensino Feevale, por ter propiciado um ambiente de aprendizagem e construção da minha formação.

RESUMO

A medição e o controle da produtividade no desenvolvimento de *software* tornam-se cada vez mais presente no dia-a-dia das empresas que estão procurando aumentar a capacidade produtiva da equipe, não só no desenvolvimento do *software*, mas também nos demais processos que fazem parte da produção do *software*, tais como, análise de requisitos, teste, documentação, implantação etc. A produtividade pode ser avaliada através de indicadores que são aplicados as métricas de estimativa de *software*, controlando a produção frente ao que foi planejado. As métricas de estimativa de *software* mais conhecidas atualmente são as de contagem de pontos por função e pontos por caso de uso. Sendo assim, este trabalho tem como objetivo desenvolver uma ferramenta que auxilie na medição e no controle da produtividade no desenvolvimento, fornecendo informações ao gerente do projeto referente às atividades de toda a equipe. Permitindo-lhe analisar através de indicadores de produtividade quais as tarefas menos produtivas, como por exemplo, retrabalho, reuniões etc., com o objetivo de buscar novos métodos para melhorar a produtividade da equipe.

Palavras-chaves: Produtividade. Desenvolvimento de *Software*. Métricas. Indicadores.

ABSTRACT

The measurement and control of productivity in software development are becoming ever more present in day-to-day businesses that are looking to increase the productive capacity of the team, not only in developing the software, but also in other processes that are part production of software, such as requirements analysis, testing, documentation, deployment. Productivity can be measured through indicators that are applied to metrics for estimation of software, managing the production opposite to what was planned. The metrics for estimation of software are best known today for the counting of function points for and Use case points. Thus, this work aims to develop a tool that assists in measuring and control of productivity in the development, providing information to the manager of the project concerning the activities of the entire team. Allowing it to examine indicators of productivity through which the less productive tasks, such as rework, meetings, aiming to seek new methods to improve the productivity of the team.

Keywords: Productivity. Software Development. Metrics. Indicators.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AIE	Arquivos de interface externos
ALI	Arquivos lógicos internos
APF	Análise de Pontos de Função
CBO	Coupling Between Objects
CE	Consulta externa
DIT	Depth of Hinheritance
EE	Entrada externa
KLOC	Milhares de linhas de código
LCOM	Lack of cohesion in Methods
LOC	Linhas de código-fonte
NOC	Number of Children
OO	Orientação a objeto
PCU	Pontos de Caso de Uso
PCU	Pontos de Caso de Uso
PF	Pontos de função
PSP	*Personal Software Process*
RFC	Response for class
SE	Saída externa
TSP	*Team* Software *Process*
UC	Caso de Uso
IFPUG	International Function Point Users

SUMÁRIO

INTRODUÇÃO

A produtividade e a qualidade no desenvolvimento de *software* sempre foram um desafio para as empresas, que buscam aumentar a produtividade da equipe sem comprometer a qualidade do *software*. É importante que as empresas utilizem medições e indicadores que possam auxiliar no controle da produtividade. No entanto, segundo Sommerville (2003), não se deve utilizar as medições de produtividade para fazer julgamentos sobre a capacidade produtiva do desenvolvedor, pois isso pode comprometer a qualidade do produto. As medições devem ser utilizadas como ferramenta para melhorar os processos de desenvolvimento de *software*, a fim de aumentar a capacidade produtiva da equipe.

Mas para avaliar a produtividade de uma equipe de desenvolvimento, primeiro é preciso conhecer quais as tarefas que fazem parte do projeto, para isto pode ser utilizado métodos de estimativa de *software* que auxiliam no planejamento das atividades que devem ser executadas para a realização do desenvolvimento do *software*. Segundo Jones (2008), a indústria de *software* tem aproximadamente 50 tipos diferentes de metodologias de desenvolvimento, e com isso tem quase tantas abordagens de métricas e medições, as mais conhecidas são:

- Pontos de caso de uso;
- Análise de Pontos de função.

Os métodos de estimativa de *software* têm um papel importante no desenvolvimento de sistema, pois a partir deles é feito o planejamento das atividades de cada integrante da equipe, ou seja, a ordem em que as tarefas devem ser executadas e também o tempo estimado para a sua realização. Utilizando-se coletas de dados e de indicadores de produtividade, podem ser realizadas comparações entre o estimado e o produzido, a capacidade produtiva de acordo com a tecnologia, etc. No entanto, o problema na indústria de *software* não está somente em utilizar a métrica de estimativa que mais se adapte ao ambiente da empresa, e sim em encontrar indicadores que auxiliem na medição da produtividade, pois muitas empresas não armazenam informações de projetos anteriores referente a produtividade de sua equipe, retrabalhos, atividades que consomem tempo do desenvolvimento, entre outras atividades.

Sendo assim, a proposta deste trabalho é desenvolver uma ferramenta para controle e medição da produtividade, com uma forma mais fácil para obter os dados das tarefas, tornando o processo de lançamento de atividades menos burocrático. Além disto, a ferramenta proposta deverá permitir ao gerente controlar a produtividade da equipe,

retrabalho, comparar tempo estimado e realizado, entre outros controles que serão abordados no decorrer do trabalho.

Com esta ferramenta pretende-se analisar quais as tarefas que consomem mais tempo do usuário, qual a capacidade produtiva de cada membro da equipe em relação a diferentes linguagens de programação, fornecendo ao gerente informações que possam auxiliá-lo na melhora dos processos e na capacitação da sua equipe.

Para o desenvolvimento deste trabalho, foram realizadas as seguintes divisões: no primeiro capítulo será abordado o tema produtividade, trazendo qual é o significado da produtividade para as empresas, quais os pontos que mais interferem na produtividade da equipe ou individual e quais os pontos que ajudam a melhorar a produtividade do desenvolvimento de *software*. Já no capítulo 2, serão apresentados os tipos de métricas de *software* mais conhecidas, como as métricas orientadas a tamanho, a função, a objeto e a pessoas. No capítulo 3, serão apresentados os métodos de estimativa de *software* de análise de pontos de função e de caso de uso. No capítulo 4, serão apresentados dois métodos de controle de produtividade individual e de equipe que foram desenvolvidos por Watts S. Humphrey, que são o PSP – *Personal Software Process* e o TSP – *Team Software Process*. Dando continuidade ao trabalho no capítulo 5, serão apresentadas as ferramentas existentes no mercado que controlam as atividades da equipe. Nos capítulos 6 e 7 serão apresentadas as necessidades, funcionalidades e a modelagem do sistema. E por último será descrito o estudo de caso e os resultados obtidos com a ferramenta.

1 PRODUTIVIDADE

O conceito de produtividade, segundo Rezende (2005), vem do latim *productivus*, que significa fértil, rendoso, proveitoso, profícuo. Que é diferente de produção, que é simplesmente quantidade produzida, sem valor e uso. Este termo admite várias interpretações, cada qual enfoca determinados objetivos e usos. Jones (1991) define que o termo "Produtividade de *software*" implica em reduzir o tempo necessário para se desenvolver novos sistemas e programas, e implica em diminuir a quantidade de dinheiro que as empresas gastam com *software*. Já Demarco (1990, p.19), define que:

> "A produtividade tem de ser definida como o benefício dividido pelo custo. O benefício é representado pelas economias em dólares e a renda do trabalho executado e os custos são o custo total, que inclui a substituição de qualquer empregado que seja utilizado para o esforço"

Mas para Rezende (2005), produtividade é a relação entre os resultados obtidos e os recursos disponíveis consumidos. Deste modo, pode-se dizer que um *software* ou sistema teve produtividade obtida quando seu resultado (produto) com qualidade foi disponibilizado no tempo predefinido, ou antes.

Como visto, para o termo produtividade de *software* existem várias definições e todas estão relacionadas em aumentar a produtividade, diminuindo custo e trazendo benefícios para os clientes. Sendo assim, a medição da produtividade para a indústria de *software* é vista como a chave para o aprimoramento da eficiência e da eficácia do desenvolvimento de *software*, pois a partir do advento das tecnologias e de técnicas de produção de *software*, a produtividade no desenvolvimento passou a ser um diferencial competitivo na indústria de *software*.

Segundo Fenton e Pfleeger (1998), a idéia de se comparar as entradas e as saídas são úteis para o desenvolvimento de *software*. Intuitivamente, o conceito de produtividade envolve o contraste entre o que entra em um processo e aquilo que é obtido na sua saída. Ele também descreve que a medida mais comum a ser utilizada é a de tamanho ao longo do esforço. Isto é, o tamanho das saídas geradas é comparado com a quantidade de esforço para transformar uma entrada em uma saída.

Entretanto, segundo Sommerville (2003), não se deve utilizar as medições de produtividade para fazer julgamentos sobre a capacidade produtiva do desenvolvedor, pois isso pode comprometer a qualidade do produto. Muitas vezes, o maior interesse fica com a

produtividade de um membro da equipe do projeto de *software*, ao invés da produtividade de todo o processo. Isto é, existe uma maior preocupação com a determinação de quão produtivos são os desenvolvedores de *software* e, assim, possa-se a tomar algumas medidas para conseguir aumentar sua produtividade, ao invés de melhorar o processo a fim de obter um melhor rendimento na produtividade.

Desta forma, deve-se prestar muita atenção ao fator humano, que é um ponto importante ao se falar em produtividade, pois todos os resultados utilizados pelos indicadores de produtividade estão relacionados com as atividades realizadas pelos envolvidos no projeto. Frente a este problema, Demarco (1990) definiu que os principais problemas das empresas não são de natureza tecnológica, mas sim sociológica. Pois não deve ser avaliado o tempo de trabalho e o produzido, e sim, outros fatores que têm influência sobre a capacidade produtiva do engenheiro de *software* como, por exemplo, ambiente de trabalho, motivação da equipe, rotatividade de pessoal, entre outros.

1.1 FATORES QUE INTERFEREM NA PRODUTIVIDADE

A maneira como as pessoas trabalham no desenvolvimento de *software*, individual ou em equipe, tem um impacto decisivo sobre os resultados obtidos. (KOSCIANSKI, 2007). Para a avaliação da produtividade é necessário conhecer as variáveis que interferem na sua medição, podendo assim melhorar e controlar estes fatores. Estas variáveis que necessitam ser observadas e analisadas acompanham a profissão desde sua criação, e atualmente tem se tornado um fator de competitividade de mercado, devido a diminuição do tempo necessário para o desenvolvimento de novos sistemas.

A empresa pode buscar no tempo ganho com a produtividade, uma forma para melhorar a qualidade do sistema, através de testes ou ainda obter uma melhora em relação a retrabalhos, que geralmente acabam sendo o gargalo da maioria dos processos. A seguir serão citados os fatores que têm influência sobre a produtividade.

1.1.1 Ambiente de trabalho

O ambiente de trabalho da empresa pode interferir muito na produtividade das pessoas, pois se o ambiente fornecido tem muitas interrupções, como o telefone tocando a todo instante ou pessoas deixando as suas atividades para conversarem assuntos que não se referem aquela tarefa, isto pode tornar um ambiente menos produtivo. Segundo Demarco (1990), são perdidos dias inteiros, e ninguém pode apontar onde eles foram. Ele também cita uma frase bastante interessante que diz o seguinte: “Existe um milhão de maneiras de se perder um dia de trabalho, mas não existe um único modo de trazê-lo de volta”.

Muitos ambientes de trabalho nas empresas de desenvolvimento são tão cheios de gente e barulhentos que os profissionais acabam se frustrando, pois não conseguem desenvolver suas tarefas com a concentração desejada. Isto ajudaria a comprovar que alguns ambientes de trabalho são produtivos para determinadas pessoas e menos produtivos para outras.

De acordo com Demarco (1990), existe um folclore entre os funcionários de desenvolvimento em todos os setores de que horas extras fazem parte da vida. Isto quer dizer que muitas tarefas não podem ser realizadas dentro do horário de trabalho e para isso são necessárias horas extras para a sua realização. Existem pessoas que preferem chegar mais cedo ou ficar depois do horário no seu local de trabalho para realizar determinadas tarefas, pois durante o dia o ambiente de trabalho é muito tumultuado e não se tem a concentração necessária para a realização da atividade.

Portanto, o ambiente de trabalho fornecido para o desenvolvedor pode trazer diversos problemas para o projeto, como por exemplo, se um desenvolvedor não consegue obter uma produtividade desejável durante o dia de trabalho e tiver que realizar a tarefa fazendo horas extras, isto irá trazer um custo maior para o projeto, ou então se o trabalho que está sendo realizado não for executado com concentração, poderá causar erros no sistema e isto acarretará em retrabalho, ainda mais se este trabalho estiver sendo repassado para uma equipe de testes e precisar retornar para ajustes. Com isso, será gerado um custo para a empresa e ainda será apontado como uma perda na produtividade. Sendo assim, o ambiente de trabalho interfere bastante na produtividade do desenvolvedor ou da equipe de projeto.

1.1.2 Motivação

Sommerville (2003) apud Maslow sugeriu que as pessoas são motivadas pela satisfação de suas necessidades, conforme Figura 1. Sendo que o nível inferior dessa hierarquia representa as necessidades de alimentação, sono etc. e a necessidade de se sentir seguras em um ambiente. Já as necessidades sociais dizem respeito à necessidade de sentir-se parte de um grupo social, ou seja, estar em contato com outras pessoas que serão consideradas um grupo social. No topo da hierarquia ficam as necessidades de estima que estão ligadas ao respeito que as outras pessoas têm por ela. E as necessidades de auto-realização estão ligadas à realização pessoal e também profissional, obtido através do seu desenvolvimento.

Figura 1 - Hierarquia das necessidades humanas.
Fonte: Sommerville (2003) apud Maslow.

Segundo Sommerville (2003), as pessoas que trabalham em organizações de desenvolvimento de *software* em geral não estão sedentas ou famintas e não se sentem fisicamente ameaçadas pelo seu ambiente. Portanto, assegurar as necessidades sociais, de estima e de auto-realização, é muito importante do ponto de vista de gerenciamento. Obter estas necessidades significa conceder às pessoas momentos de encontro com os colegas ou reconhecer o trabalho realizado, mostrando que elas são reconhecidas pela organização, assim se estaria atendendo as necessidades sociais e de estima. As necessidades de auto-realização podem ser proporcionadas, atribuindo as pessoas tarefas difíceis mais não impossíveis de se realizar ou então concedê-las responsabilidades pelo seu próprio trabalho.

Em um estudo psicológico realizado os profissionais de desenvolvimento foram classificados em três tipos. (SOMMERVILLE (2003) apud BASS E DUNTEMAN):

Orientados a tarefas, que são motivados pelo trabalho que fazem. Na engenharia de *software*, eles são os técnicos motivados pelo desafio intelectual do desenvolvimento de *software*

Auto-orientados, que são principalmente motivados pelo sucesso pessoal e pelo reconhecimento. Eles estão interessados no desenvolvimento de *software* como meio de atingir seus próprios objetivos.

Orientados a interações, que são motivados pela presença e pelas ações dos colegas de trabalho. À medida que o desenvolvimento de *software* se torna mais centrado no usuário, os indivíduos orientados a interações estão se tornando cada vez mais envolvidos.

No entanto, a motivação de uma pessoa não está somente ligada às necessidades pessoais, mas também está ligada à motivação do grupo ou da equipe de trabalho em que estão inseridas. Segundo Demarco (1990), o propósito de uma equipe não é a consecução do objetivo, mais sim o alinhamento do objetivo. Ou seja, quando a equipe está coesa e mais integrada a realização dos objetivos fica mais eficiente, já que a equipe está focada na realização de um objetivo comum. Além disto, a coesão da equipe tem algumas vantagens tais como, desenvolver um padrão de qualidade para o grupo, o grupo trabalha mais estreitamente em conjunto, os membros da equipe conhecem o trabalho desenvolvido por cada indivíduo e a programação sem egoísmo pode ser praticada.

Todos os aspectos que foram observados são remetidos a motivação de cada indivíduo, pois existe uma “competição” saudável entre os integrantes da equipe que elevam as necessidades pessoais de estima e auto-realização, onde as dificuldades são resolvidas em equipe e todo o desenvolvimento pode ser realizado em pares. Este tipo de gerenciamento do desenvolvimento pode trazer grandes resultados, já que o trabalho realizado por uma pessoa será revisado pelo seu colega, isso irá afetar significativamente a produtividade, já que os defeitos podem ser resolvidos ainda na programação ou ainda antes de serem implementados. Com isso, os problemas serão resolvidos antes de serem repassados para a equipe de testes.

Estes grupos ou equipes geralmente são mais motivados, pois estão sempre em comunicação e buscado a satisfação da realização dos objetivos, além de ser uma equipe com baixa rotatividade de pessoal, já que as pessoas mais motivadas e se sentido parte de um grupo não estão tão preocupadas com salários e com o seu reconhecimento. Além de que rotatividade de pessoal é outro problema que será abordado no próximo tópico.

1.2 FATORES QUE AUXILIAM NA PRODUTIVIDADE

Como visto anteriormente, existem fatores que interferem na produtividade, mas também existem fatores que evoluíram e contribuem para no aumento da produtividade. Desde o inicio da profissão de desenvolvimento de *software* muitas atividades foram modificadas, muitos processos foram melhorados tornando o desenvolvimento de *software* uma tarefa além das linguagens de programação, ferramentas foram criadas para a automatização de processos, entre outras melhorias obtidas através dos anos.

1.2.1 Melhores linguagens de programação

As linguagens de programação sofreram enormes modificações desde os anos 50, quando os programadores criavam programas de computador codificando laboriosamente os 0s e 1s que o *hardware* executa. (YOURDON, 1990, p. 135). Após a primeira geração de linguagens de máquina, vieram as linguagens de montagem nos anos 60. Nos anos 70 chegou a terceira geração, as linguagens procedurais, que são as mais conhecidas atualmente, como COBOL, FORTRAM, PASCAL, C, entre outras. A partir da quarta geração de linguagens, que foram uma evolução da terceira transferindo o seu enfoque, os programadores deixaram de se preocupar com os confusos detalhes, consumidores de tempo, de edição e validação das entradas. (YOURDON, 1990, p. 136).

Estas novas linguagens trouxeram argumentos de que se pode aumentar a produtividade, segundo Jones (1991), quanto mais alto o nível da linguagem, menor a quantidade de código que deve ser escrita para complementar uma dada função. Entretanto, como a programação representa de 10 a 15% do projeto geral de desenvolvimento de sistema, o ganho total em produtividade é muitas vezes bem menos substancial. (YOURDON, 1990, p. 136). No entanto, as linguagens de programação são extremamente importantes, pois elas irão afetar a forma de resolver ou analisar o problema, mas não irão trazer um grande ganho na produtividade, devido a parte de implementação não ser a maior parte no desenvolvimento de um sistema.

1.2.2 Ferramentas que automatizam o desenvolvimento de sistemas

O motivo de um problema de produtividade é que boa parte do trabalho de desenvolvimento de um sistema automatizado é, ironicamente, executado de forma manual. (YOURDON, 1990, p. 137). Esta é uma constatação que pode ser verificada em muitas organizações que não criaram suas próprias ferramentas de automatização. Porém, segundo Jones (1991), os geradores de programas se originaram na década de 1960, no campo da geração de compiladores, que foi uma das primeiras subdisciplinas da ciência da computação. Mas já na década de 80, os geradores começaram a ser no campo de aplicações comerciais e atualmente já existem ferramentas que automatizam os testes de *software*, e ajudam a manter diagramas de fluxos de dados, definições de requisitos entre outras atividades. Existem ferramentas que, a partir da especificação de requisitos e definições de regras de sistemas, são capazes de gerar o código fonte eliminando o trabalho de codificação do sistema. Com isso, será possível ter um ganho significativo no desenvolvimento, já que poderá ser utilizado este tempo em outras áreas da produção do *software*.

1.2.3 Controle de engenharia de *software*

Outra forma identificada para aumentar a produtividade é através das coleções de ferramentas, técnicas e controles geralmente conhecida como engenharia de *software*. (YOURDON, 1990, p. 137). Estes controles são geralmente métodos de estimativa de *software*, análises estruturadas, utilizando caso de usos, diagramas de seqüência, entre outros métodos. Segundo Yourdon (1990), as técnicas estruturadas de desenvolvimento de sistemas tem tido um modesto impacto sobre a produtividade, entre 10 a 20%, sobre a produtividade dos profissionais de desenvolvimento de sistemas durante a fase de desenvolvimento do projeto. Entretanto, os sistemas desenvolvidos com a utilização das técnicas estruturadas têm geralmente custo inferior de manutenção e confiabilidade maior.

2 MÉTRICAS

Segundo Pressman (2002), medir é fundamental em qualquer disciplina de engenharia. Ao medir a produtividade ou a qualidade do *software*, preocupa-se em obter dados históricos, como por exemplo: Qual foi a produtividade no desenvolvimento de *software*? Qual foi a qualidade do *software* produzido?

As métricas de *software* possibilitam a avaliação da eficácia dos projetos de *software*. Com as métricas coletadas em projetos e em etapas do processo, é possível construir um conjunto de indicadores que ao longo do tempo podem ser aperfeiçoados. Estes indicadores podem ser quantidades de erros encontrados durante os testes, tempo para desenvolvimento por pontos de função ou pontos por caso de uso, entre outros indicadores que serão identificados nos próximos capítulos.

Segundo Koscianski (2007), o propósito das coletas não deve ser punir erros, mas aprimorar estimativas de prazos e custo melhorando o gerenciamento do projeto. Além disso, as informações obtidas servem para a avaliação individual ou da equipe, podendo assim melhorar o treinamento ou a comunicação dentro do projeto. Koscianski (2007) também cita a importância dos desenvolvedores fornecerem dados realísticos, evitando alterá-los para justificar suas dificuldades. Para isto, a empresa deve fornecer *feed-back*[1] constante à equipe e também individualmente, estabelecendo um ambiente de confiança e honestidade.

Segundo Vazquez (2003), com a utilização de métricas, justificar e defender as decisões tomadas fica mais fácil, pois, o gerente do projeto não toma decisões com base nas suas experiências, mas sim com utilização de avaliação de indicares que refletem uma tendência futura, considerando que estes foram definidos previamente. Em função disto, é necessário manter informações passadas de outros projetos e não somente do projeto atual. No entanto, é preciso definir objetivos específicos e para cada objetivo deve-se gerar um grupo de perguntas, com isso, a partir de algumas respostas é possível identificar métricas que possa quantificar as resposta. A figura 2 demonstra o ciclo que precisa ser estabelecido para encontrar novos indicadores e aperfeiçoar os anteriores.

---

[1] *Feed-Back*: É o procedimento que consiste no provimento de informação à uma pessoa sobre o desempenho, conduta ou eventualidade executada por ela e objetiva reprimir, reorientar e/ou estimular uma ou mais ações determinadas, executadas anteriormente.

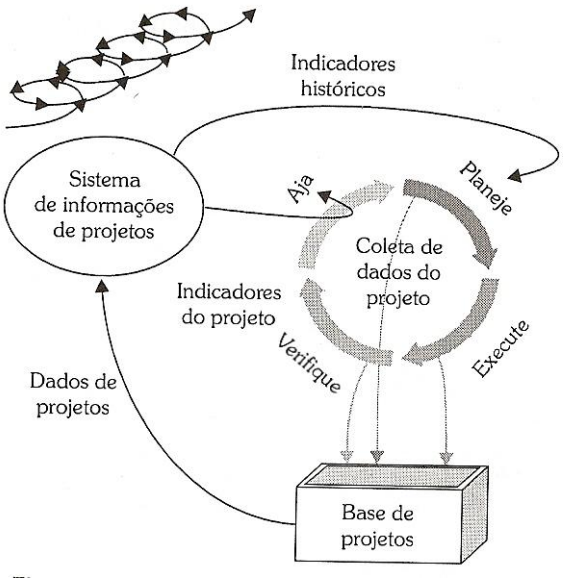


Figura 2 - Fundamento para melhoria continua.
Fonte: Vazquez, 2003, p 23.

As utilizações de métricas de *software* são muito importantes, pois somente através do controle e da medição é possível melhorar a qualidade dos produtos e processos de desenvolvimento do *software*. Portanto, é necessário que as métricas sejam estabelecidas no início do projeto, já na fase de planejamento onde são realizadas as estimativas de custo, tempo, riscos, dentre outros. No entanto, para a definição destas métricas que irão acompanhar o projeto, auxiliando o gerente a melhor administrar as pessoas e os processos, deve ser observado qual o tipo de métrica que melhor se enquadra ao desenvolvimento, já que existem métricas orientadas a tamanho, a função ou a objetos. Estas métricas podem ser utilizadas vinculadas com métricas de produtividade, qualidade e técnica, conforme mostra a figura 3.

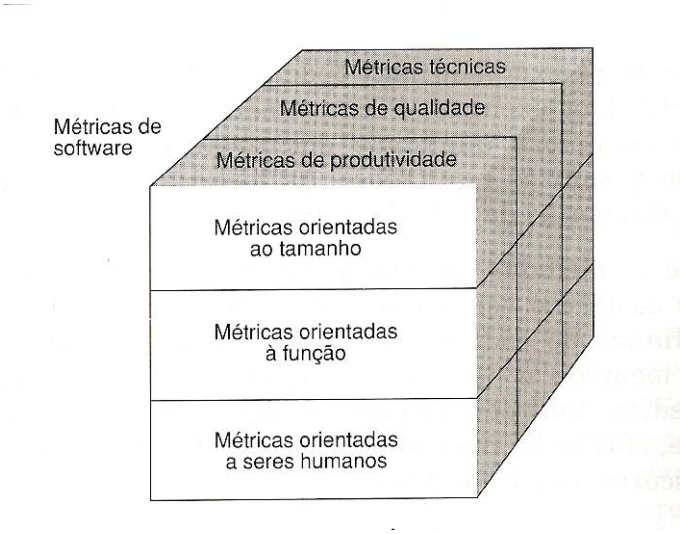


Figura 3 - Divisão das métricas em categorias.
Fonte: Pressman, 2002, p 62.

2.1 MÉTRICAS DE *SOFTWARE* ORIENTADAS A TAMANHO

As métricas de *software* orientadas a tamanho são derivadas das medidas diretas do *software*, como por exemplo, linha de código-fonte (LOC), milhares de linhas de código (KLOC). A utilização das métricas de LOC e suas variações são bastante discutidas, considerando que com o advento das linguagens de alto nível este parâmetro passou a ser bastante penalizado. Esta métrica, ao ser utilizada em linguagens de alto nível, pode orientar o gerenciamento da produtividade em uma direção contrária e não contribuindo para o controle da produtividade. (JONES, 1991).

Ao se utilizar a medida de linhas de código-fonte produzidas por unidade de tempo em linguagens de alto nível, deve-se prestar atenção, pois a métrica poderá causar uma taxa de produção de linhas de código menores e não maiores. Para a utilização de linha de código como um indicador de produtividade, devem ser observados alguns itens, para não incorrer na utilização de dois pesos e duas medidas, deixando de contar ou então contando determinadas linhas, como por exemplo:

- Linha de comentários ou linha em branco;
- Diretrizes de compilação;
- A contagem de múltiplos comandos ou declarações em uma única linha como várias linhas, uma para cada comando ou declaração;

- A contagem de uma única linha nos casos em que um único comando ou declaração é expresso em múltiplas linhas;
- A desconsideração de delimitadores de blocos de comandos nos casos em que sua utilização seja opcional.

Como pode ser observado, apesar de o método ser bastante intuitivo, pode-se conter uma série de regras em sua contagem, e para isto não existe um padrão definido. Além de que, segundo Vazquez (2003), para o cliente de um sistema saber que ele possui 1.000.000 LOC é o mesmo que dizer que ele possui 4520 *bitstuffs.*

Segundo Jones (1991), este movimento retrógrado é diretamente proporcional ao nível da linguagem e as linguagens de mais alto nível terão as taxas de produção mais baixas quando um ciclo de desenvolvimento completo for medido. No entanto, LOC foi uma métrica bastante utilizada até meados de 1970. A partir daí surgiram diversas linguagens de programação e conseqüentemente a necessidade de outras formas de estimar o tamanho de *software*.

2.2 MÉTRICAS DE *SOFTWARE* ORIENTADAS A FUNÇÃO

Até o início da década de 1970, a principal forma de medir um *software* era utilizando LOC. A simplicidade da medida resulta ao mesmo tempo em seu baixo custo e precisão. (KOSCIANSKI 2007). A medição funcional é um termo geral para métodos de dimensionamento de *software* baseado em funções requeridas pelos usuários.

A métrica de análise de pontos de função foi desenvolvida na década de 1970 na IBM, por Allan Albrecht, ao ser solicitado a ele que fosse medida a produtividade de vários projetos de *software* desenvolvidos pela IBM. Estes projetos tinham sido desenvolvidos em uma grande variedade de linguagens, tornado inviável uma análise conjunta da produtividade. (VAZQUES, 2003). A utilização desta técnica no Brasil começou significativamente no início da década de 1990, com um forte apoio de uma grande empresa, Unisys[2]. Porém, o interesse do mercado consolidou-se apenas quando grandes contratos públicos começaram a ser baseados em pontos de função. (VAZQUES, 2003).

[2] A Unisys é uma empresa mundial de Tecnologia da Informação.

Como uma forma de medir *software* considerando as funcionalidades criadas, a medida pode ser aplicada antes de o programa ser escrito, baseando-se na definição de requisitos do sistema, permitindo estimar o esforço e o cronograma para o desenvolvimento das atividades. Além disso, a métrica está bastante difundida através de pontos de função e também em pontos de caso de uso, pois elas são independentes de linguagem de programação utilizada no desenvolvimento de softwares.

2.3 MÉTRICAS DE *SOFTWARE* ORIENTADAS A OBJETO

Algumas métricas foram definidas especialmente para *software* orientado a objeto. Segundo, Fenton e Pfleeger (1998), com o aumento da popularidade da orientação a objeto, foi desenvolvido métodos para analisar a medida das estruturas orientadas a objeto.

As métricas de *software* orientadas a objeto que serão descritas nesta seção foram especificadas nas bibliografias de Koscianski (2007) e Fenton e Pfleeger (1998) que são métodos baseados em Chidamber e Kemerer (1994).

- A profundidade de herança (DIT – *Depth of Hinheritance Tree*) mede a distância, ao longo da árvore de herança entre uma classe qualquer e a raiz da árvore. Quanto maior é o valor de profundidade da classe, mais métodos foram herdados e maior será a complexidade ligada ao funcionamento do código;
- O número de filho (NOC – *Number of Children*) mede o número de classe diretamente derivadas a partir de uma dada classe. Neste caso, se o valor de profundidade for elevado indica a re-utilização de código;
- O acoplamento (CBO – *Coupling Between Objects*) de uma classe A para uma classe B é dado pelo número de métodos de B chamados a partir de A, adicionando ao número de atributos B referenciados por A;
- A resposta para uma classe (RFC – *Response For Class*) é o número de métodos que pode ser invocado em resposta a uma mensagem enviada por um objeto. Quanto maior for o valor da métrica, maior a complexidade de funcionamento. Como conseqüência, provavelmente mais teste devem ser realizados para assegurar a ausência de defeitos;
- A falta de coesão (LCOM – *Lack of cohesion in Methods*) é definida como segue:
  - $\{M_1, M_2 \dots M_n\}$ = Conjunto de métodos da classe;

- I_k = Conjunto de variáveis de instâncias utilizadas por M_k;
- P = {(I_i, I_j) | não há interação entre I_i e I_j};
- Q ={(I_i, I_j) | há interação entre I_i e I_j};
- Se P > Q, o valor da métrica é dado por P-Q; zero caso contrário.

Um valor elevado desta métrica pode indicar a oportunidade de dividir a classe em subclasses. A subdivisão é um princípio importante para tratar a complexidade.

Para as métricas OO, outros autores propuseram outras formas, conforme Koscianski (2007 apud Genero, 2000) definiu uma técnica muito interessante que consiste em investigar intuitivamente o acoplamento e a coesão. Isso pode ser feito utilizando gráficos que ilustra ao avaliador a complexidade estrutural do produto e permitem facilmente identificar gargalos.

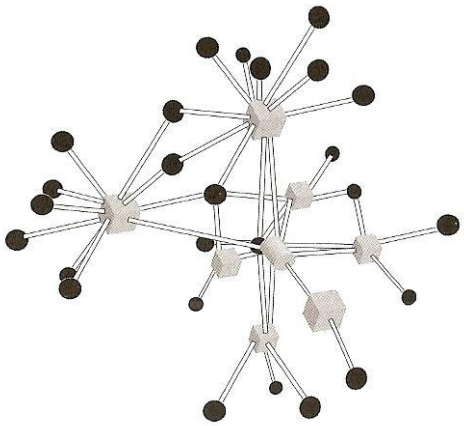


Figura 4 - Visualização da estrutura de uma classe.
Fonte: KOSCIANSKI, 2007, p 242.

2.4 MÉTRICAS DE *SOFTWARE* ORIENTADAS A PESSOAS

As métricas de *software* orientadas a pessoa, são obtidas a partir da definição de tarefas e alocações de recursos realizados durante o projeto. (KOSCIANSKI, 2007). Estas métricas podem ser empregadas para conhecer o desempenho individual ou de toda a equipe, o que permite estimativas mais seguras de recursos alocados nos projetos. A utilização pelo gerente dos relatórios de acompanhamento das atividades e recursos alocados serve para identificar o desempenho individual ou de toda a equipe, permitindo-lhe estimar a duração de atividades em projetos futuros.

3 CONTAGEM DE PONTOS DE FUNÇÃO E CASO DE USO

Nos primórdios da computação o custo de *software* compreendia uma pequena parte do custo global do *software*. Hoje, o *software* é o elemento mais caro de muitos sistemas baseados em computador. Devido a isso, um erro na estimativa de custo e esforço pode fazer a grande diferença entre o lucro e o prejuízo. As estimativas de custo e de esforço de *software* jamais serão uma ciência exata, já que muitas variáveis, tais como, humanas, técnicas, ambientais e políticas podem afetar o custo final do *software* e do esforço aplicado para desenvolvê-lo. (PRESSMAN, 2002).

A estimativa de projetos de *software* é uma forma de resolução de problemas e, na maioria dos casos, o problema a ser resolvido é muito complexo e para isso deve ser dividido em partes. Com isso, o problema é decomposto em partes menores, sendo mais fácil de gerenciá-los. Alcançar estimativas efetivas de tamanho do *software* utiliza uma técnica de analogia simples. A técnica de analogia é aplicada comparando o projeto atual com dados de projetos similares passados. A partir do conhecimento do tamanho de um projeto similar, estima-se o tamanho do novo projeto em nódulos menores e estimar o percentual do tamanho de cada módulo em relação ao projeto passado. Então, o tamanho total do projeto é obtido através da soma do tamanho estimado para cada módulo. (Vasquez, 2003).

O método de estimativa de *software* é um processo que deixa muitas incertezas, pois é feito a partir de cálculos realizados em cima de artefatos de documentação. Neste capítulo, será abordo as duas técnicas mais conhecidas de estimativa de *software*: Análise de Pontos de função e Pontos por caso de uso.

3.1 ANÁLISE DE PONTOS DE FUNÇÃO

A contagem de pontos de função (PF) é realizada através da análise das funcionalidades do projeto de *software*, providas a partir dos dados de entrada e saída de softwares externos. Para a realização da contagem de PF é preciso avaliar todas as etapas do processo conforme exibido na figura 5, que apresenta as etapas do processo de contagem, bem como suas relações de interdependência. (VAZQUEZ, 2003).

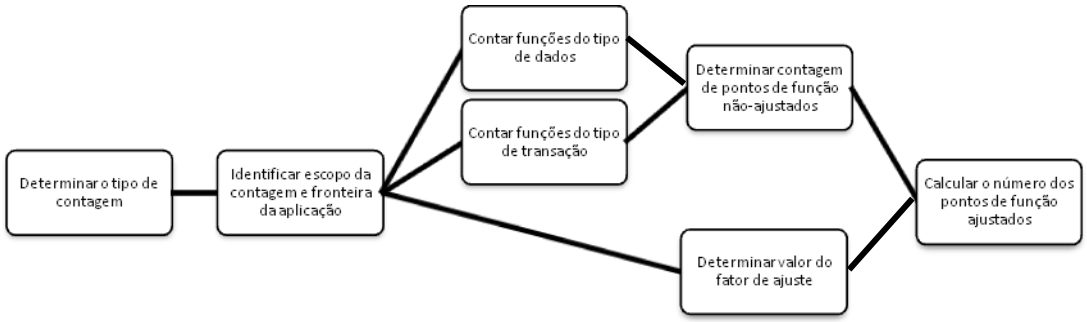


Figura 5 - Visão geral do processo de contagem de pontos de função.
Fonte: Vazquez, 2003.

3.1.1 Determinação do tipo de contagem

A determinação do tipo de contagem pode estar ligada tanto a projetos como a aplicações. Para determinar o tipo de contagem existem três tipos que podem ser verificados na Figura 6 que mostra o relacionamento existente entre os tipos.

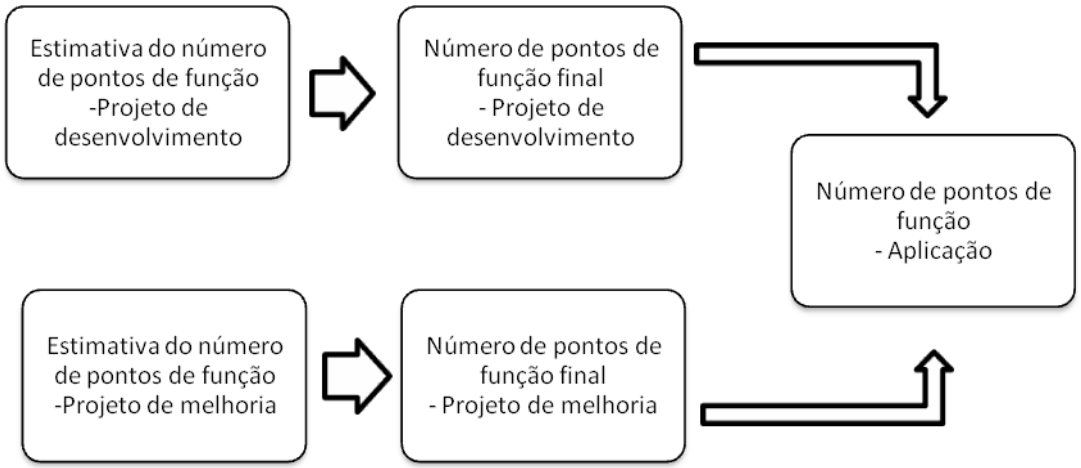


Figura 6 - Relacionamento entre os tipos de contagem.
Fonte: Vazquez, 2003.

3.1.1.1 Contagem de um projeto de desenvolvimento

O número de pontos de função de um projeto de desenvolvimento mede a funcionalidade fornecida aos usuários finais do *software* quando da sua primeira instalação. Isso significa que essa contagem também abrange as eventuais funções de conversão de dados necessárias à implementação do sistema. (VAZQUEZ, 2003)

3.1.1.2 Contagem de um projeto de melhoria

O número de pontos de função de um projeto de melhoria mede as funções adicionais, modificadas ou excluídas do sistema pelo projeto, e também as eventuais funções de conversão de dados. (VAZQUEZ, 2003).

3.1.1.3 Contagem de uma aplicação[3]

O número de pontos de função de uma aplicação mede a funcionalidade fornecida aos usuários por uma aplicação instalada. Também é chamado de número de pontos de função instalados por uma aplicação. Esse número fornece uma medida da atual funcionalidade obtida pelo usuário da aplicação. Ele é inicializado ao final da contagem de números de pontos de função do projeto de desenvolvimento, sendo atualizado no término do projeto de melhoria que altera a funcionalidade da aplicação. (VAZQUEZ, 2003).

---

3 Na definição do IFPUG: “uma aplicação é um conjunto coeso de dados e procedimentos automatizados que suportam um objetivo de negócio, podendo consistir de um ou mais componentes, módulos ou subsistemas. Freqüentemente, o termo “aplicação” é utilizado como sinônimo para sistema, sistema aplicativo ou sistema de informação”.

3.1.2 Fronteira da aplicação

A fronteira da aplicação é a interface conceitual que delimita o *software* que será medido e o mundo exterior. Quando há mais de uma aplicação incluída no escopo de uma contagem, várias fronteiras devem ser identificadas. (VAZQUEZ, 2003).

O IFPUG especifica as seguintes regras para a determinação da fronteira da aplicação:

- Sua determinação deve ser feita com base no ponto de vista do usuário. O foco deve estar no que ele pode entender e descrever;
- A fronteira entre aplicação deve ser baseada na separação das funções conforme estabelecido pelos processos do negócio, não em considerações tecnológicas;
- Em projetos de melhoria, a fronteira estabelecida no início do projeto deve estar de acordo com a fronteira já estabelecida para a aplicação sendo modificada.

A identificação das fronteiras é um passo importante no processo de contagem, pois um posicionamento incorreto da fronteira pode alterar a perspectiva de uma visão lógica. Este processo muitas vezes é difícil devido a não conseguir delinear onde uma aplicação começa e a outra termina.

3.1.3 Funções do tipo dados

As funções do tipo dados representam as funcionalidades fornecidas pelo sistema ao usuário, para atender a suas necessidades de dados. (VAZQUEZ, 2003).

Arquivos lógicos internos (ALI): grupo de dados persistentes, relacionados logicamente, identificável pelo usuário, mantidos pelo sistema e alimentados por entradas externas ou valores calculados.

Arquivos de interface externos (AIE): grupo de dados persistentes, relacionados logicamente, identificável pelos usuários, porém utilizados apenas para consultas do sistema. Os dados são mantidos e alimentados por outras aplicações.

A complexidade dos arquivos lógicos internos e externos é determinada com a relação de baixa, média e alta conforme a quantidade de tipos de dados e tipos de registro. Como demonstra a tabela 1.

Tabela 1 - Tabela de complexidade funcional dos ALI e AIE.

	< 20	20 – 50	>50
1	Baixa	Baixa	Média
2 - 5	Baixa	Média	Alta
> 5	Média	Alta	Alta

Fonte: Vazquez, 2003.

Após a determinação da complexidade dos arquivos, é possível calcular a quantidade de pontos referentes ao arquivo, conforme exibido na tabela 2.

Tabela 2 - Tabela de pontos de funções dos tipos de dados.

Tipo de função	Baixa	Média	Alta
ALI	7 PF	10 PF	15 PF
AIE	5 PF	7 PF	10 PF

Fonte: Vazquez, 2003.

3.1.4 Funções do tipo transação

As funções do tipo transações representam as funcionalidades de processamento de dados fornecidos pelo sistema do usuário. (VAZQUEZ, 2003).

Entrada externa (EE): Um processo lógico em que os dados são introduzidos no sistema. Estes dados podem ser informações de controle ou de negócios. As entradas externas representam o fluxo de informações que adentram o sistema, tendo com principal intenção manter um ou mais arquivos lógicos internos ou alterar o comportamento do sistema.

Saídas externas (SE): Um processo lógico em que os dados enviados ao exterior da fronteira do sistema, por exemplo, na emissão de relatórios de totais de faturamento por cliente. Sua principal intenção é apresentar informação ao usuário através de lógica de processamento que não seja apenas uma simples recuperação de dados ou informações de controle.

Consulta externa (CE): Um processo lógico que envolve um par consulta resposta. Os dados são recuperados exclusivamente de arquivos internos e interfaces externas. Nenhum arquivo lógico é alterado neste processo.

A complexidade das transações de entrada externa, saída externa e consulta externa deve ser classificada com relação à sua complexidade funcional, baseado no número de arquivos referenciados e no número de tipo de dado. Após determinar a quantidade dos

arquivos referenciados e dos tipos de dados, a complexidade é estabelecida de acordo com a tabela 3 e a tabela 4.

Tabela 3 - Tabela de complexidade funcional para entradas externas

Arquivos referenciados	Tipos de dados		
	< 5	5 - 15	> 15
< 2	Baixa	Baixa	Média
2	Baixa	Média	Alta
> 2	Média	Alta	Alta

Fonte: Vazquez, 2003.

Tabela 4 - Tabela de complexidade funcional para saídas externas e consultas externas.

Arquivos referenciados	Tipos de dados		
	< 6	6 – 19	> 19
< 2	Baixa	Baixa	Média
2 - 3	Baixa	Média	Alta
> 3	Média	Alta	Alta

Fonte: Vazquez, 2003.

Após a determinação da complexidade das funções de tipos de transação, é possível calcular a quantidade de pontos referentes ao arquivo, conforme exibido na tabela 5.

Tabela 5 - Tabela de pontos de funções dos tipos funções de transações.

Tipo de função	Baixa	Média	Alta
EE	3 PF	4 PF	6 PF
SE	4 PF	5 PF	7 PF
CE	3 PF	4 PF	6 PF

Fonte: Vazquez, 2003.

3.1.5 Pontos de função não-ajustados

Cada função do tipo dado e transação são classificadas com a relação à sua complexidade em baixa, média e alta.

3.1.6 Fator de ajuste

O valor de fator de ajuste é baseado em 14 características gerais de sistemas, conforme tabela 6. Para cada característica é atribuído um nível de influência sobre o sistema que pode variar em um intervalo discreto de zero a cinco, conforme tabela 7.

Tabela 6 - Características gerais de sistema.

	Características
1	Comunicação de dados
2	Processamento distribuído
3	Performance
4	Configuração altamente utilizada
5	Volume de transações
6	Entrada de dados on-line
7	Eficiência do usuário final
8	Atualização on-line
9	Complexidade de processamento
10	Reutilização
11	Facilidade de instalação
12	Facilidade de operação
13	Múltiplos locais
14	Facilidade de mudança

Fonte: Vazquez, 2003.

Tabela 7 - Nível de influência.

	Nível de influência
0	Nenhuma influência
1	Influência mínima
2	Influência moderada
3	Influência média
4	Influência significativa
5	Grande influência

Fonte: Vazquez, 2003.

3.1.7 Cálculos dos pontos de função

O cálculo dos pontos de função ajustados é dividido nos três tipos de contagem: Projeto de desenvolvimento, projeto de melhoria e aplicação.

3.1.7.1 Projeto de desenvolvimento

Para o cálculo de pontos de função para projeto de desenvolvimento foi definida a seguinte fórmula:

DFP = (UFP + CFP) x VAF

Em que:

- DFP: Número de pontos de função do projeto de desenvolvimento;
- UFP: Número de pontos de função não-ajustados das funções disponíveis após a instalação;
- CFP: Número de pontos de função não-ajustados das funções de conversão;
- VAF: Valor do fator de ajuste.

3.1.7.2 Projeto de melhoria

Para o cálculo de pontos de função para projeto de melhoria foi definida a seguinte fórmula:

EFP = [(ADD + CHGA + CFP) x VAFA] + (DEL x VAFB)

Em que:

- EFP: Número de pontos de função do projeto de melhoria;
- ADD: Número de pontos de função não-ajustados das funções incluídas pelo projeto de melhoria;
- CHGA: Número de pontos de função não-ajustados das funções modificadas. Reflete funções depois das modificações;
- CFP: Número de pontos de função não-ajustados adicionados pela conversão;
- VAFA: Valor do fator de ajuste depois do projeto de melhoria;
- DEL: Número de pontos de função não-ajustados das excluídas pelo projeto de melhoria;
- VAFB: Valor do fator de ajuste antes do projeto de melhoria.

3.1.7.3 Aplicação

Para o cálculo de pontos de função para a aplicação existem duas fórmulas. Uma para a primeira contagem dos pontos de função da aplicação e a outra para recalcular seu tamanho após um projeto de melhoria ter alterado suas funcionalidades.

Para a função de contagem inicial foi definida a seguinte fórmula:

AFP = ADD x VAF

Em que:

- AFP: Número de pontos de função ajustados da aplicação;
- ADD: Número de pontos de função não-ajustados das funções instaladas;
- VAF: Valor do fator de ajustes da aplicação;

Já a fórmula para a contagem de pontos de função após o projeto de melhoria é definida com a seguinte fórmula:

AFP = [(UFPB + ADD + CHGA) - (CHGB - DEL)] x VAFA

Em que:

- AFP: Número de pontos de função ajustados da aplicação;
- UFPB: Pontos de função não-ajustados da aplicação antes do projeto de melhoria;
- ADD: Pontos de função não-ajustados das funções incluídas pelo projeto de melhoria;
- CHGA: Pontos de função não-ajustados das funções alterados pelo projeto de melhoria depois do seu término;
- CHGB: Pontos de função não-ajustados das funções alterados pelo projeto de melhoria antes do seu término;
- DEL: Pontos de função não-ajustados das funções excluídas pelo projeto de melhoria;
- VAFA: Valor do fator de ajuste depois do projeto de melhoria.

3.2 PONTOS POR CASO DE USO

A técnica de estimativa por Pontos de Caso de Uso foi proposta em 1993 por Gustav Karner, da *Objectory* (hoje, *Rational Software*). Ela baseia-se no método de análise de pontos de função APF e também em Mark II[4], bastante utilizada na Inglaterra. O diferencial da métrica por Casos de Uso é a forma como é lançada a estimativa, já que o método trata de estimar o tamanho de um sistema de acordo com o modo como os usuários o utilizarão, a complexidade de ações requerida por cada tipo de usuário e uma análise em alto nível dos passos necessários para a realização de cada tarefa, em um nível muito mais abstrato que a técnica de APF.

3.2.1 Contagem dos pontos de caso de uso - PCU

A análise de sistemas orientados a objetos normalmente utiliza os diagramas de caso de uso para descrever as funcionalidades do sistema. Sendo assim, é possível estimar-se o tamanho do *software* baseando-se em um conjunto simples de métricas e modificadores, similar à técnica de APF. Mas é importante frisar que o grau de detalhe do caso de uso analisado influencia diretamente na qualidade da medição. Por isso, deve haver um cuidado ao analisar os casos de uso e medir somente os que estão em nível de sistema.

O método de contagem dos Pontos de casos de uso é descrito em seis etapas, que foram definidas por Gustav Karner. (MEDEIROS, 2004).

[4]Mark II: A estimativa de tamanho é baseada nas funções do software de acordo com a visão do usuário.

3.2.1.1 Contar os autores e atribuir o grau de complexidade

Relacionar os atores, classificá-los de acordo com seu nível de complexidade (simples, médio ou complexo) atribuindo respectivamente os pesos 1, 2 ou 3, conforme a tabele 8.

Tabela 8- Classificação de atores.

Complexidade do autor	Descrição	Peso
Simples	Outro sistema acessado através de uma API de programação	1
Média	Outro sistema interagindo através de um protocolo de comunicação, como TCP/IP ou FTP	2
Complexo	Um usuário interagindo através de uma interface gráfica (stand-alone ou Web)	3

Fonte: Medeiros, 2004.

3.2.1.2 Calcular o total de Pesos não ajustado dos atores

O cálculo do total de pesos não ajustados do atores, conhecido como TPNAA, se dá a partir da contagem do número de atores em cada categoria. Em seguida deve-se multiplicar pelo fator de peso elegido para cada categoria conforme Tabela 8.

3.2.1.3 Contar os caso de usos e atribuir o grau de complexidade

Contar os casos de uso e atribuir o grau de complexidade sendo a complexidade baseada no número de classes e transações conforme tabela 9.

Tabela 9 - Classificação dos casos de uso

Tipo de caso de uso	Descrição	Peso
Simples	Considerar até 3 transações com menos de 5 classes de análise	5
Médio	Considerar de 4 a 7 transações com 5 a 10 classes de análise	10
Complexo	Considerar de 7 transações com pelo menos de 10 classes de análise	15

Fonte: Medeiros, 2004.

3.2.1.4 Calcular o total de pesos não ajustados por caso de uso

Calcular o total de pesos não ajustados dos casos de uso, conhecido como TPNAUC, é realizado através da multiplicação da quantidade de caso de uso pelo respectivo peso definido na categoria, conforme a Tabela 9.

3.2.1.5 Calcular o total de pontos de casos de uso não ajustáveis

Calcular PCU´s não ajustados, também chamados de PCUNA, de acordo com a seguinte fórmula:

PCUMA = TPNAA + TPNAUC

3.2.1.6 Determinar o fator de complexidade técnica

Os fatores de complexidade técnica variam em uma escala de 0 a 5, de acordo com o grau de dificuldade do sistema a ser construído. O valor 0 indica que a grau não está presente ou não é influente, 3 influência média e o valor 5 indica influência significativa através de todo o processo. Após determinar o valor dos fatores, multiplicar pelo respectivo peso ilustrado na tabela 10, somar o total e aplicar a seguinte fórmula:

Fator de complexidade técnica (FCT) = 0.6 + (0.01 * Somatório do Fator técnico).

Tabela 10 - Fatores de complexidade técnica.

	Descrição	Peso
1	Sistemas distribuídos	2,0
2	Desempenho da aplicação	1,0
3	Eficiência do usuário final (on-line)	1,0
4	Processamento interno complexo	1,0
5	Reusabilidade do código em outras aplicações	1,0
6	Facilidade de instalação	0,5
7	Usabilidade (Facilidade operacional)	0,5
8	Portabilidade	2,0
9	Facilidade de manutenção	1,0
10	Concorrência	1,0
11	Características especiais de segurança	1,0
12	Acesso direto para terceiros	1,0
13	Facilidades especiais de treinamento	1,0

Fonte: Medeiros, 2004.

3.2.1.7 Determinar a influência do fator ambiental

Os fatores de complexidade ambientais indicam a eficiência do projeto e estão relacionados ao nível de experiência dos profissionais. Esses fatores descritos na tabela 11 são determinados através da escala de 0 a 5, onde 0 indica baixa experiência, 3 indica média experiência e 5 indica alta experiência. Após determinar o valor de cada fator, multiplicar pelo peso e somar o total dos valores. Em seguida, aplicar a seguinte fórmula:

Fator de complexidade ambiental (FCA) = 1,4 + (-0,03 * Somatório do Fator Ambiental).

Tabela 11 - Classificação de fatores ambientais.

	Descrição	Peso
1	Familiaridade com o Processo de Desenvolvimento de *Software*	1,5
2	Experiência na Aplicação	0,5
3	Experiência com OO, na Linguagem e na Técnica de Desenvolvimento	1,0
4	Capacidade do Líder de Projeto	0,5
5	Motivação	1,0
6	Requisitos estáveis	2,0
7	Trabalhadores com dedicação parcial	-1,0
8	Dificuldade da Linguagem de Programação	-1,0

Fonte: Medeiros, 2004.

3.2.1.8 Calcular os pontos de caso de uso ajustados

Esse cálculo é realizado com base na multiplicação dos PCU não ajustados, na complexidade técnica e na complexidade ambiental através da seguinte fórmula:

PCUA = PCUNA * Fator de complexidade técnica * Fator de complexidade ambiental

3.2.2 Considerações sobre pontos de caso de uso

Gustav Karner, ainda sugere que podemos estimar a produtividade calculando-se 20 horas de trabalho por caso de uso. Com base nos recursos utilizados pelos projetos de seu estudo, tentou ainda encontrar a correlação entre PCU e os recursos necessários para desenvolver o projeto através da adoção da regressão linear, mas seus dados não foram suficientes para encontrar a relação proposta. (ANDRADE, 2004)

Andrade (2004, apud SCHNEIDER e WINTERS, 2001) sugere um refinamento na técnica de Karner: a técnica sugere que a presença de certos atributos influencia diretamente a média de horas por ponto calculado, utilizando a seguinte lógica:

- Conta-se a quantidade de fatores técnicos entre 1 e 6 que receberam nível de influência maior que 3;
- Soma-se o valor obtido à quantidade de fatores ambientais entre 7 e 8 que receberam valor de influência menor que 3.

O somatório indica a quantidade sugerida de horas por ponto de caso de uso a ser adotada no projeto, sendo a média sugerida de:

- 20 horas por ponto para um resultado de 2 ou menor;
- 28 horas por ponto caso o somatório resulte em 3 ou 4;
- 36 horas por ponto para valores maiores que 4.

Neste último caso, os autores da técnica sugerem que o projeto seja revisto: talvez haja a necessidade de treinamento de pessoal, adequação de tecnologia ou revisão de requisitos, para garantir um melhor aproveitamento de recursos e redução no cronograma previsto.

4 CONTROLE DE PRODUTIVIDADE

Controlar a produtividade de desenvolvimento *software* deve ser um processo contínuo. Devem ser mantidos os dados históricos de projetos anteriores sobre a produtividade individual e também da equipe. Estes dados serão utilizados na busca de novos indicadores de produtividade e também no aperfeiçoamento dos indicadores já existentes.

Neste capítulo, será descrito os processos de controle e planejamento de desenvolvimento de *software* utilizando como as bases o PSP (*Personal Software Process*) e TSP (*Team Software Process*). Estes dois processos foram descritos por Watts S. Humphrey no início da década de 1990, onde se iniciou utilizando o PSP como parte do processo de implantação do CMM. Estes métodos serão importantes para o trabalho, pois irão trazer alguns dos indicadores que serão utilizados na proposta de desenvolvimento da ferramenta de controle e medição da produtividade, por considerar o desenvolvimento individual e da equipe.

4.1 PSP – PERSONAL SOFTWARE PROCESS

Personal Software Process (PSP) é um processo de desenvolvimento de *software* projetado para ser utilizado por engenheiros de *software* para a elaboração de projetos individuais, foi desenvolvido por Watts Humphrey no início da década de 1990, para orientar o planejamento e o desenvolvimento. O foco do PSP é na melhoria do processo do indivíduo, tornando sua forma de trabalho mais disciplinada. Para isto é utilizado uma conjunto de métodos, formulários e roteiros para planejar, medir e gerenciar o trabalho do indivíduo.

O PSP tem como filosofia que é preciso que cada um tenha responsabilidades pela qualidade do trabalho, para isto é preciso compreender as próprias habilidades e aplicá-las sobre as tarefas para gerir as fraquezas e construir sobre os pontos fortes. Segundo Koscianski (2007), para realizar um bom trabalho de forma rápida é preciso que os desenvolvedores façam um planejamento prévio de suas atividades. Esse planejamento, como as demais tarefas, deve seguir um processo bem definido. Para entender seu desempenho pessoal, o programador precisa medir o tempo gasto com cada etapa, os defeitos encontrados e removidos e o tamanho do *software*. O PSP é dividido em três níveis:

- Nível 0 (PSP0): Fundamentos de medidas e formatos de relatórios;
- Nível 1 (PSP1): Planejamento e estimativas de tamanho e tempo;
- Nível 2 (PSP2): Controle pessoal de qualidade de projeto;
- Nível 3 (PSP3): Extensão a projetos grandes.

4.1.1 Estrutura do PSP

A estrutura do PSP, segundo Koscianski (2007), baseia-se nos requisitos do *software*, faz-se o planejamento utilizando um conjunto de scripts que guiam esse trabalho. Durante a execução de cada fase (projeto, codificação, etc.) deve-se armazenar dados sobre tempo e defeitos nos logs. Na última fase chamada de pós-morte, todos os dados são sintetizados em um resumo do planejamento que é liberado junto com o produto final, conforme mostra a figura 7.

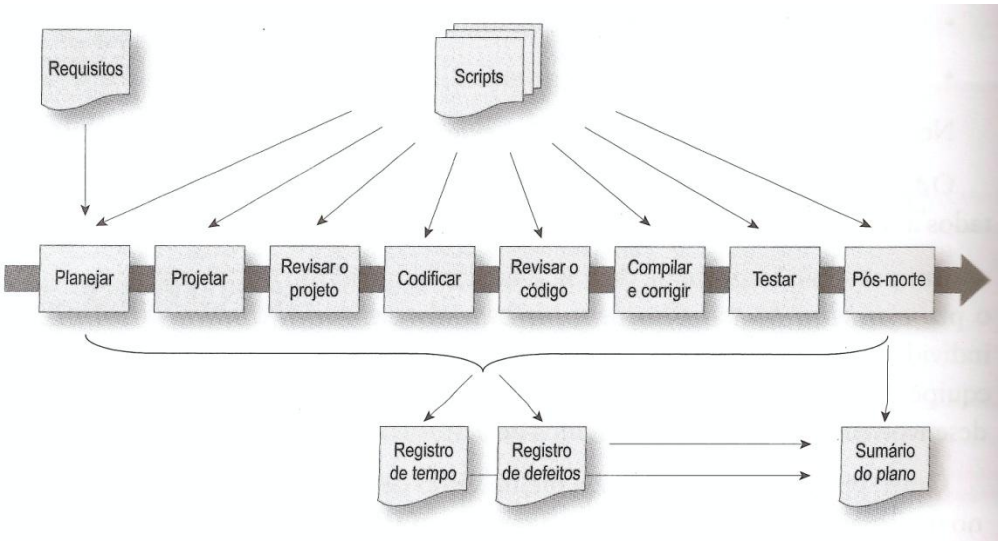


Figura 7 - Fluxo de processos do PSP.
Fonte: Koscianski, 2007.

No modelo sugerido pelo PSP o desenvolvedor deve realiza uma seqüência de atividades claramente determinadas. Portanto, em cada fase apresentada na figura 7 as atividades devem ser bem definidas, o que pode ser feito por meio de scripts, conforme a tabela 12. Esta tabela de scripts está definida dentro do nível 1 do PSP de acordo com Watts Humphrey.

Tabela 12 - Script do PSP nível 1.

Número	Fase	Descrição
-	Critérios de entrada	Descrição do programa Formulário de plano de projeto Modelo de estimativa de tamanho Dados históricos de projetos anteriores Padrão para classificar tipos de defeitos
1	Planejamento	Produzir ou obter uma declaração de requisitos Estimar os tamanhos mínimos, médio e máximo de tamanho do programa e de tempos de desenvolvimento Preencher um diário (*log*) de registros de tempo
2	Projeto	Projetar o programa sendo um formato Preencher um *log* de registro de tempo
3	Codificação	Implementar, usando um formato padrão de codificação Preencher um *log* de registro de tempo
4	Revisão	Revisar o código fonte seguindo o roteiro específico Preencher um *log* de registro de tempo
5	Compilação	Compilar o programa Corrigir e anotar os defeitos Preencher um *log* de registro de tempo
6	Teste	Testar o programa Corrigir e anotar cada defeito encontrado Preencher um *log* de registro de tempo
7	Pós-morte	Completar o formulário de plano do projeto com os dados de tempo, tamanho e defeitos encontrados Revisar os dados de defeito e alterar o *cheklist* de revisão de código Preencher um *log* de registro de tempo
-	Critérios de saída	Programa inteiramente testado Um formulário de plano de projeto completo, com dados de conclusão preenchidos Logs de registros de tempo e defeitos preenchidos

Fonte: KOSCIANSKI, 2007.

4.1.2 Plano de projeto

Todas as atividades, como visto na tabela 10, devem ser definidas previamente, para isto foi definido um formulário padrão. Este formulário foi definido sendo o plano de projeto, que define o que será feito e como o trabalho será realizado, conforme tabela 13. Alguns dados do plano são a definição de cada tarefa que será executada e as estimativas de tempo e de recursos necessários. Entretanto, este plano será utilizado como uma ferramenta para o controle gerencial, permitindo que durante o projeto possa-se comparar o que foi estimado e o que realmente foi realizado.

Tabela 13 - Plano de projeto no PSP (KOSCIANSKI, 2007)

	Planejado	Realizado	Até o momento
Resumo			
Minutos/Loc			
Loc/Hora			
Defeitos/KLoc			
Yield			
A/F-r			
Tamanho do programa			
Novo e alterado			
Tamanho máximo			
Tamanho mínimo			

	Planejado	Realizado	Até o momento	% Até o momento
Tempo (minutos)				
Planejamento				
Projeto				
Codificação				
Revisão do código				
Compilação				
Teste				
Pós-morte				
Total				
Tempo total mínimo				
Tempo total máximo				
Defeitos inseridos				
Planejamento				
Projeto				
Codificação				
Revisão do código				
Compilação				
Teste				
Total				
Defeitos removidos				
Planejamento				
Projeto				
Codificação				
Revisão do código				
Compilação				
Teste				
Total				

Fonte: KOSCIANSKI, 2007.

O plano de projeto do PSP é desenvolvido utilizando como base linhas de código, porém neste trabalho será utilizado pontos de caso de uso, por ser um método que foi desenvolvido para sistemas orientados a objetos, conforme descrito no capítulo 3.

O preenchimento do plano de projeto inicia com a informação do que foi **planejado**, que pode ser obtido através da estimativa de *software*, no capítulo 3 foram descritas duas

formas de estimativa de *software*. Ao final da implementação de todo o projeto, será preenchida a coluna **realizado**, mas este resultado também será obtido ao se completar cada etapa do roteiro do PSP.

Com o decorrer do desenvolvimento de vários programas, o programador obtém a informação para coluna **Até o momento,** que é obtido através dos dados históricos dos outros programas. Já a coluna **% Até o momento,** é utilizada para contabilizar proporções em relação aos valores históricos.

Portanto, para a realização da estimativa de tempo são anotadas a hora de início e término de cada atividade, incluindo também as interrupções para atender o telefone ou conversar com outros membros da equipe ou reuniões, todas estas pausas devem ser anotadas e consideradas, mas apenas o tempo realmente gasto com a codificação será contado. Assim sendo, a estimativa de tempo deve ser realizada em cada fase do projeto.

Também é preciso identificar os defeitos encontrados e para isso o PSP define que defeito é tudo que esteja errado no *software*, como erros de arquitetura, na representação de diagramas, problemas de algoritmos, entre outros. Os erros são registrados em uma tabela específica para defeitos, conforme Tabela 14.

Tabela 14 - Log para registro de defeitos.

Data	Número	Tipo	Inserido	Removido	T_remover	Origem
Descrição						

Fonte: KOSCIANSKI, 2007.

Nesta tabela o campo **número** representa o identificador de cada defeito encontrado, ele deve ser seqüencial. Já o **tipo** é preenchido de acordo com a tabela 15. O campo **inserido,** deve contar a fase em que o defeito foi encontrado e o campo **removido** a fase em que foi solucionado o defeito. Em **T_Remover,** deve ser anotado o tempo gasto para remover o defeito. No campo **origem,** é preenchido com "X" se o defeito não for por conseqüência de outro, caso contrário o número identificado do defeito original é escrito neste campo. Já no último campo o desenvolver descreve rapidamente o defeito.

Tabela 15 - Tipos de defeitos de *software*.

Código	Nome	Exemplos
10	Documentação	Requisitos, diagramas
20	Sintaxe	Erros de sintaxe no código
30	Construção (build)	Versões de bibliotecas
50	Interface	Omissões, falhas de projetos

Fonte: KOSCIANSKI, 2007.

4.1.3 Considerações sobre PSP

O PSP é um método de controle que procura melhorar e disciplinar o individuo, tendo como objetivo aprimorar o desempenho do desenvolvedor e a qualidade do produto desenvolvido. Porém, a utilização do PSP é bastante complicada, já que o desenvolvedor deve utilizar parte de seu tempo para coletar e analisar as métricas. Outro ponto a ser considerado é que o PSP foi desenvolvido para o processo individual de desenvolvimento. No entanto, para a utilização de método em equipe foi desenvolvido o TSP (*Team software Process*) que será descrito a seguir.

4.2 TSP – TEAM SOFTWARE PROCESS

Team Software Process, ou simplesmente, TSP é um *framework* para produção de *software*, que enfatiza o processo, o produto, o trabalho em equipe, dando destaque ao planejamento e gerenciamento do projeto. Segundo Humphrey (1999), o desenvolvimento está dividido em oito fases, que estão definidas como Lançamento da Equipe do Projeto, Desenvolvimento da Estratégia, Desenvolvimento do Plano, Definição dos Requisitos, Design de Alto Nível, Implementação, Integração e Teste do Sistema e Postmortem. Ele ainda oferece métodos prontos para serem utilizados no processo de forma que os mesmos não precisam ser reinventados ou descobertos e possam, portanto, ser reutilizados, diminuindo assim o tempo do projeto.

Sendo assim o TSP baseia-se no PSP utilizando as medidas de tamanho, tempo e defeito, mas ele ainda adiciona a data de termino das tarefas. Para todas as medidas e projeções os dados são coletados individualmente, e ainda são analisados semanalmente pela equipe para terem uma idéia do andamento do projeto frente ao plano de projeto.

4.2.1 Estrutura

Segundo Koscianski (2007) o primeiro requisito para aplicações do TSP é que cada membro da equipe tenha sido treinado no PSP. Este é o primeiro passo para monta uma

equipe de trabalho no TSP. Logo após, o TSP adiciona elementos necessários para que se possa realizar o trabalho em equipe, como mostra a figura 8. Portanto, todos os membros da equipe devem trabalhar para executar uma determinada tarefa, que no TSP as tarefas são denominados lançamentos do time.

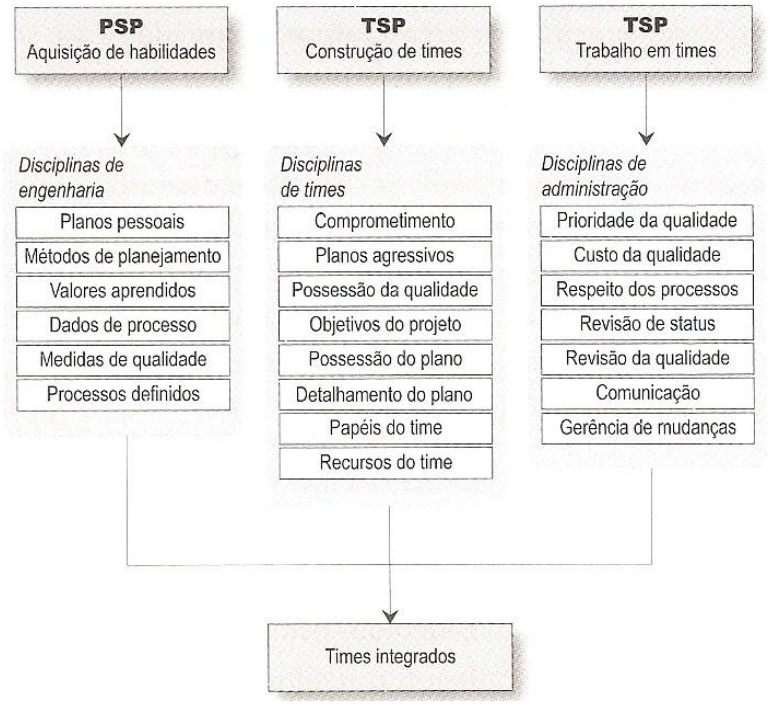


Figura 8 - Estrutura do TSP.
Fonte: KOSCIANSKI, 2007.

4.2.2 Fase do processo

Como descrito anteriormente o processo do TSP é compostos por oito fases, que são definidas por Humphrey (1999) como: lançamento da equipe, desenvolvimento da estratégia, desenvolvimento do plano, definindo os requisitos, projeto de alto nível, implementação, testes de integração e de sistema e *Postmorten*.

No entanto, nesta seção somente serão descritas as fases de lançamento da equipe, desenvolvimento da estratégia, desenvolvimento do plano e de implementação, pois para o objetivo do trabalho estas fases são mais importantes, pois tratam de como é feito toda a distribuição e definição dos papeis dentro da equipe.

4.2.2.1 Lançamento da equipe

O Lançamento da Equipe compreende a fase inicial do processo TSP, na qual são discutidos vários temas durante os quatro dias de duração desta fase. Segundo Koscianski, (2007), o processo de lançamento é composto por uma série de atividades de planejamento do trabalho a ser realizado. O trabalho a ser realizado em cada um dos dias está descrito na tabela 16.

Tabela 16 - Processo de lançamento do TSP.

Dia 1	**Dia 2**	**Dia 3**	**Dia 4**
Estabelecimento dos objetivos do produto e dos negócios.	Construção de plano *top-down* e de próxima fase	Procedimentos de estimativa de risco	Revisão gerencial
Atribuição de papeis e definição dos objetivos do time	Desenvolvimento do plano de qualidade	Lançamentos de relatórios	Execução do lançamento de pós-morte
Produção de uma estratégia de desenvolvimento	Construção de plano *Button-up* e planos balanceados		Revisão do processo

Fonte: KOSCIANSKI, 2007.

É no lançamento da Equipe que também deve ser discutido os objetivos gerais e comuns da equipe. Estes objetivos devem ser objetivos ousados, porém realistas e alcançáveis para que haja estímulo e os mesmos não sejam deixados de lado ou tornem-se desestimulantes.

4.2.2.2 Desenvolvimento da estratégia

Após o lançamento da equipe é preciso que a mesma desenvolva uma estratégia de trabalho. Esta etapa tem como principal objetivo minimizar o risco do desenvolvimento exceder o tempo programado para a conclusão do projeto. Para isso Humphrey (1999), definiu os seguintes passos para que o objetivo descrito acima seja alcançado.

- No primeiro ciclo de desenvolvimento deve-se produzir um produto executável com as funções mínimas do produto final;

- Este produto, resultante do primeiro ciclo, deve ser uma base para o restante do produto, que possa, facilmente, ser estendia;
- Em todos os ciclos ser um produto com uma alta qualidade, permitindo que o mesmo possa ser facilmente testado;
- Ter uma arquitetura modular para que membros da equipe possam trabalhar independentemente.

4.2.2.3 Desenvolvimento do plano

Segundo Humphrey (1999), o planejamento ajuda os engenheiros da equipe a trabalhar de maneira mais eficiente, com mais produtividade e sem desperdício de tempo. Além disso, algumas atividades podem ser esquecidas se não forem planejadas. Portanto, planejar um projeto não é uma tarefa muito fácil, pois projetos grandes e complexos consomem muitas horas de planejamento.

Com a utilização do TSP é possível analisar a sobrecarga de tarefas de alguns membros da equipe enquanto outros estão mais "folgados". Conforme Humphrey (1999), isto faz com que aqueles que não estão sobrecarregados tenham que, em alguns casos, esperar pela conclusão do trabalho daqueles sobrecarregados, atrasando assim o grupo. Para resolver este problema TSP propõe que nesta fase de planejamento seja feito um balanceamento de trabalho entre os membros da equipe.

4.2.2.4 Implementação

Nesta fase, está definida a etapa de construção das partes do produto e conseqüentemente o produto como um todo. O desenvolvimento de cada parte projetada na fase anterior será, preferencialmente, realizado por quem a projetou. Porém, mesmo que o engenheiro que a projetou seja o mais indicado a programá-la, isto pode ser diferente, pois a estratégia que será adotada não é imposta pelo TSP.

4.2.3 Considerações sobre TSP

O TSP foi desenvolvido para integrar os métodos que são utilizados pelo PSP, a fim de melhorar os processos da equipe. Pois em muitas empresas é difícil uma equipe de desenvolvimento de *software*, conseguir desenvolver um *software* com qualidade, com um baixo grau de defeitos e ainda dentro do prazo programado, maximizando os lucros e obtendo uma produtividade desejável. *Team Software Process* é um completo e detalhado processo que pode ser adotado por organizações que desejam melhorar seus processos de desenvolvimento de *software*. Porém, o TSP, assim como a maioria dos outros processos, deve ser customizado e adaptado a realidade de cada organização.

Koscianski (2207) descreve como exemplo de utilização do TSP a empresa Teradyne onde a taxa média de defeitos era de 20/KLOC, e com primeiro projeto utilizando TSP apresentou uma taxa de defeitos de 1 Defeito/KLOC.

5 ANÁLISE DE FERRAMENTAS EXISTENTES NO MERCADO

Para desenvolver a ferramenta de controle e medição de produtividade, viu-se a necessidade de uma pesquisa no mercado para verificar as ferramentas já existentes. Portanto, este capítulo tem o objetivo de descrever as funcionalidades e os aspectos positivos e negativos de três ferramentas existentes no mercado. Cabe ressaltar que esta análise irá apoiar o desenvolvimento da ferramenta de medição e controle de produtividade proposta no início deste trabalho e não avaliar as ferramentas.

Atualmente no mercado existe diversas ferramenta que controlam as tarefas de uma equipe de desenvolvimento de *software*, no entanto foram selecionadas três ferramentas devido a algumas características específicas que serão descritas no decorrer deste capítulo. Todos os sistemas são *softwares* proprietários, necessitando de licença para versões completas. Todos possuem versão de avaliação, por este motivo escolhidos para teste.

5.1 JIRA

JIRA é desenvolvida pela empresa australiana Altassian, que é especializada em ferramentas de colaboração e desenvolvimento. A empresa foi fundada em 2002 pelos então saídos da universidade, Farquhar Scott e Mike Cannon-Brookes. Conforme o site da empresa, a ferramenta é utilizada por mais de onze mil e duzentas organizações em mais de cento e sete países, tendo clientes empresas em vários segmentos e portes, tais como, Boeing, HSBC, Nokia, 3M, BMW, etc.

O *software* desenvolvido pela empresa tem como finalidade principal o gerenciamento de projetos e acompanhamento de tarefas e erros. A ferramenta é desenvolvida sobre a plataforma Java e está disponível a partir de três versões: *Standard, Professional, Enterprise*. O *software* está disponível em duas modalidades, sendo uma disponível através da internet e a outro para instalação local na empresa.

Na tela inicial do sistema é possível verificar as tarefas pendentes para o usuário e as tarefas que estão em andamento, conforme figura 9. Estas informações na tela inicial são importantes, pois facilitam o acesso as tarefas a serem realizadas. A navegação do *software*

pode ser realizado através do menu na parte superior da tela ou também através dos acessos a partir dos itens pendentes.

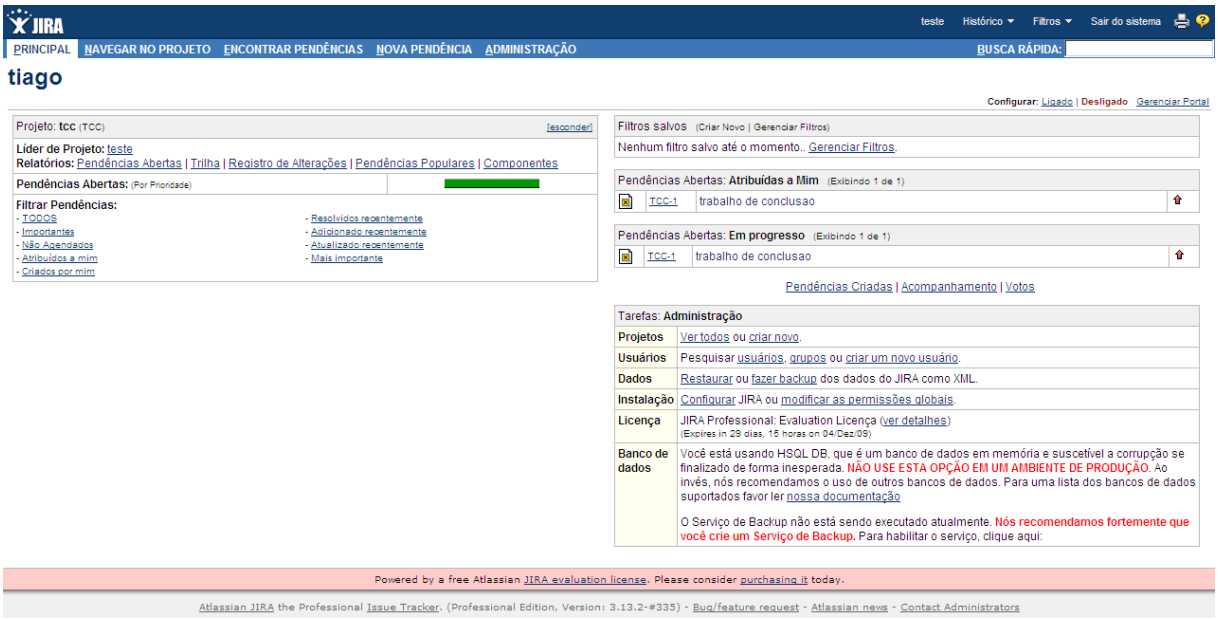


Figura 9 - Tela inicial do JIRA.
Fonte: http://www.atlassian.com/software/jira

A ferramenta, também tem uma tela para gerenciamento do projeto onde é possível gerenciar o projeto, adicionar pendências, ver as estatísticas, quantidade de pendências abertas, por membro e relatórios com agrupamentos por responsável, componentes, projeto, prioridades, entre outros. Além disso, o lançamento das pendências para o projeto é realizado a partir do menu superior em nova pendência ou a partir da tela do projeto em criar nova pendência. Os campos para entrada dos dados da tarefa são customizáveis de acordo com a necessidade da empresa ou do projeto, porém a ferramenta já vem com os campos padrões, ou seja, descrição, data de término, critérios de urgência, entre outros.

Para os lançamentos das atividades é necessário acessar a tela de gerenciamento da tarefa, conforme figura 12. Para iniciar a atividade é preciso clicar no *link* iniciar progresso no menu da direita, conforme desta na figura 12.

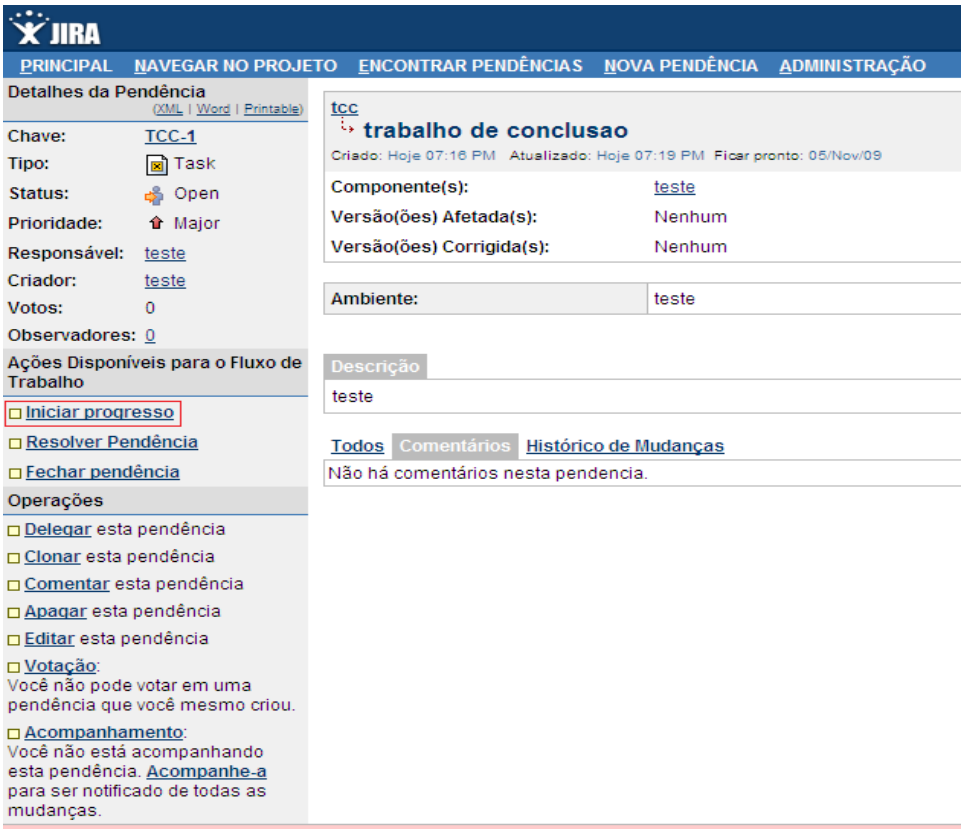


Figura 10 - Tela de gerenciamento da tarefa – JIRA.
Fonte: http://www.atlassian.com/software/jira

Na tela de gerenciamento da tarefa é possível realizar outras ações como: Concluir a pendência, acompanhar o seu andamento, encaminhar para outra pessoa, entre outras ações conforme pode ser observado na figura 10.

Os relatórios do *software* são customizados de acordo com a necessidade, sendo as gerações dinâmicas a partir dos filtros definidos pelo usuário do sistema. Entretanto, a ferramenta apresentada não fornece informação para o controle da produtividade, apenas gerencia os projetos e também as tarefas da equipe.

5.2 ACEPROJECT

AceProject foi desenvolvido pela empresa Websystems inc. de Québec, Canadá que foi fundada em 2001. O *software* foi desenvolvido pela empresa com o intuito de gerenciar projetos e atividades, através da internet. Conforme o site da empresa que tem como cliente

empresas de pequeno, médio e grande porte, tais como Toyota, Goodyer, ADP. O *software* é licenciado através de quatro modalidades: *Basic, Standard, Advanced e Gold*.

Na figura 11, é possível visualizar as tarefas no projeto, acessando no menu da esquerda é possível visualizar as tarefas incompletas e as completas, relatório de status do projeto, gráfico de *Grantt*, entre outras informações.

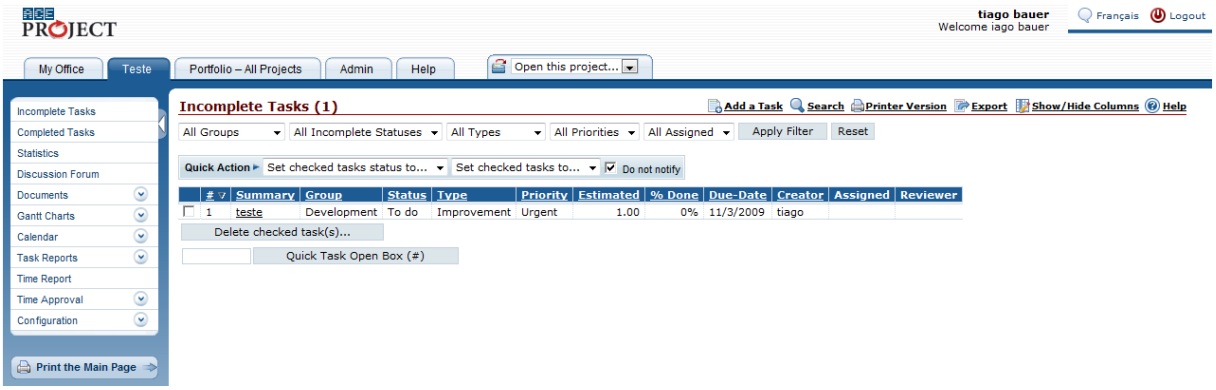


Figura 11 - Tela de gerenciamento do projeto – AceProject.
Fonte: http://www.aceproject.com

A ferramenta tem uma tela para cadastro de tarefas do projeto, onde é possível informar, tais como prioridade, data de início, data de conclusão, entre outras informações. Além disso, o lançamento das atividades pode ser realizado através da tela de edição da tarefa, ou através do menu inicial, conforme figura 12, onde pode ser observado no canto esquerdo superior um link com o nome *Close IN/OUT* (destaque em vermelho) ou atreves de um menu na tela inicial do sistema. Também é possível observar que o lançamento das atividades está direitamente relacionado a uma tarefa de um projeto, não sendo possível identificar outras atividades não relacionadas a tarefa.

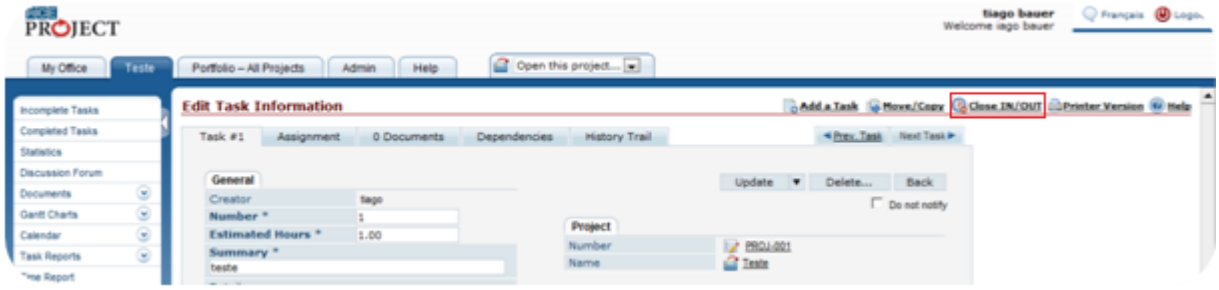


Figura 12 - Tela de gerenciamento da tarefa – AceProject.
Fonte: http://www.aceproject.com

O *software* também fornece relatórios de estatísticas, conforme figura 13, onde pode ser analisado os dados em nível de tarefas por grupo, tipos de tarefas, estados das tarefas e

prioridades das tarefas. Em todos estes agrupamentos é informado o tempo estimado, tempo realizado, quantidades de tarefas, percentual de médio estimado e percentual médio realizado.

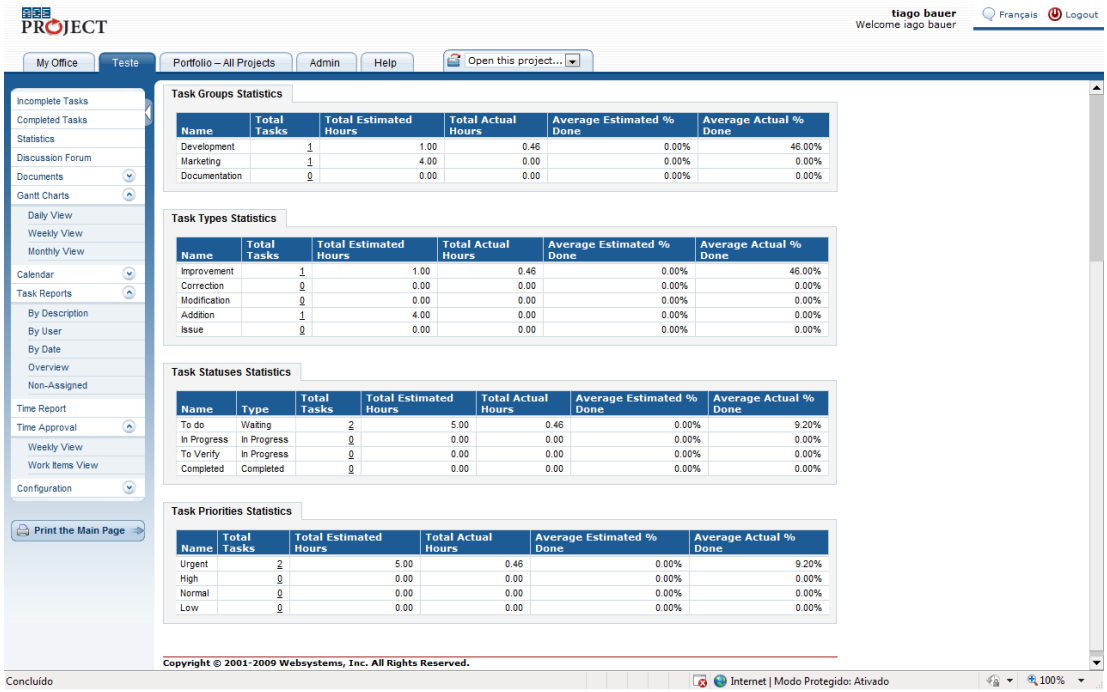


Figura 13 - Tela de estatísticas – AceProject.
Fonte: http://www.aceproject.com

O *software* ainda fornece outros relatórios com estes mesmos agrupamentos, só que com as informações detalhadas. No entanto, não serão detalhados estes relatórios por não terem relevância para o trabalho.

5.3 CLICKTIME

O ClickTime é desenvolvido pela empresa ClickTime.Com, que iniciou suas atividades sendo uma divisão da Mann Consulting, localizada em San Francisco no ano de 1997. No final de 1999 a empresa foi tirada da organização se tornando uma única empresa. O *software* desenvolvido pela empresa tem como finalidade controlar os tempos gastos com tarefas de projetos. É uma ferramenta onde o gerenciamento é feito através de um site e os lançamentos do tempo trabalhado é realizado a partir de um módulo desktop que é instalado localmente para cada usuário. Segundo o site da empresa ela tem usuários em mais de 30 países, sendo alguns destes clientes empresas como Johnson & Johnson, Google, Xerox Canadá, HP ING, entre outras. A comercialização da ferramenta é realizada através de duas modalidades basic e

corporate, sendo o valor calculado a partir do número de funcionários que irão utilizar o sistema.

A utilização do sistema é através de abas, conforme a figura 20. Sendo a primeira aba para o gerenciamento pessoal, onde são exibidas as abas para gerenciamento de tarefas por dia, por semana, controle de gastos, relatórios etc. Já a segunda aba contém informações sobre gerenciamento da empresa, onde podem ser lançados os clientes, projetos, tarefas, visualizar os relatórios dos projetos, tempos estimados, configurações preferenciais etc.

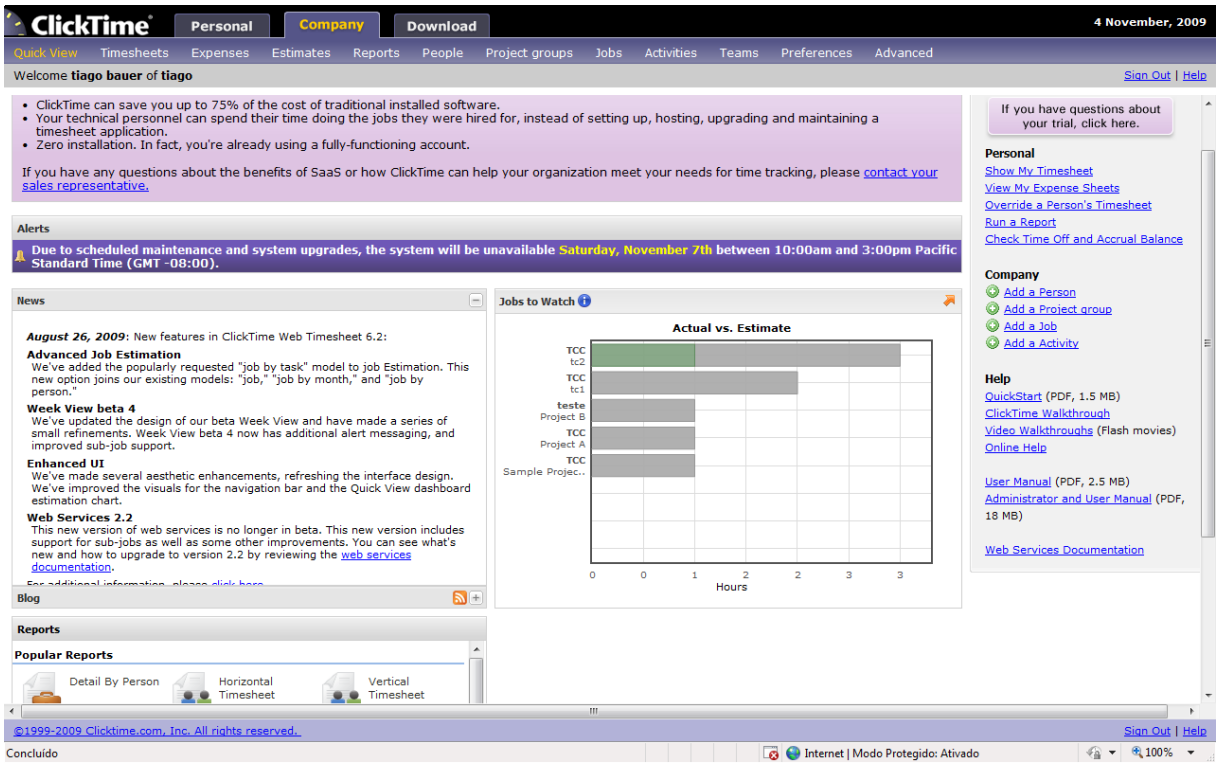


Figura 14 - Tela inicial do sistema ClickTime
Fonte: http://www.clicktime.com

A Ferramenta conta uma tela para lançamentos das tarefas, sendo que nesta ferramenta foi possível identificar que o controle da tarefa é realizado através de tempo estimado que é informado na tarefa. Além disso, o lançamento de tempo para a realização das atividades é realizado por um modulo que é utilizado localmente no computador do usuário, sendo uma versão desktop, conforme exibido na figura 15 é possível selecionar o projeto e a tarefa e clicando no ícone amarelo no final da linha da tarefa é possível iniciar o controle do tempo.

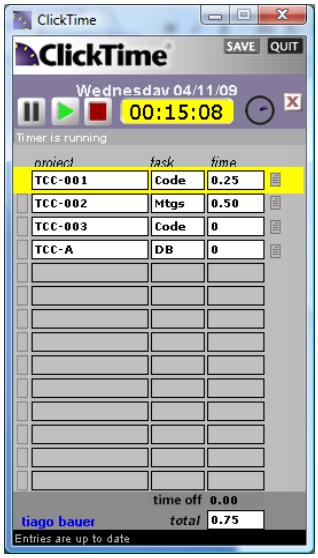
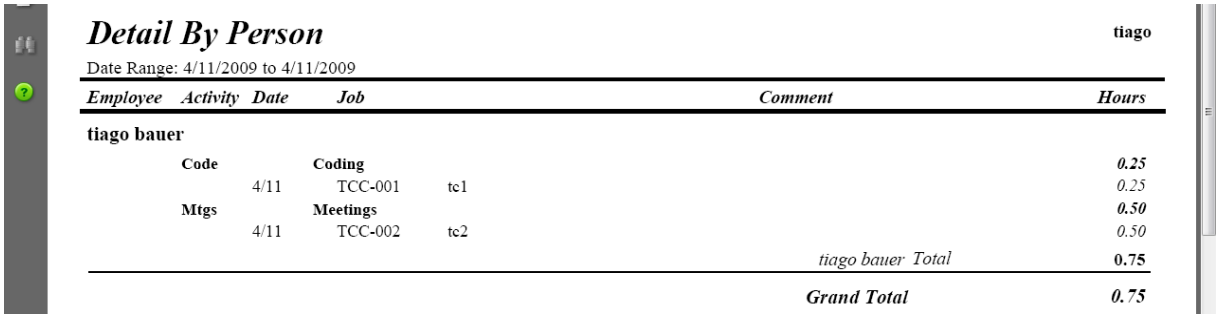


Figura 15 - Tela de lançamento de atividades – ClickTime
Fonte: http://www.clicktime.com

O *software* gera os relatórios agrupados em seis categorias, sendo elas pessoas, atividades, diversos, tarefas, projetos e custos. Neste trabalho não será detalhado todos os relatórios fornecidos pelo *software*, mas chamo a atenção para dois relatórios. Um agrupa as informações por usuário, atividade e tarefa, conforme figura 16.



Detail By Person **tiago**

Date Range: 4/11/2009 to 4/11/2009

Employee	*Activity*	*Date*	*Job*		*Comment*	*Hours*
tiago bauer						
	Code		**Coding**			***0.25***
		4/11	TCC-001	tc1		*0.25*
	Mtgs		**Meetings**			***0.50***
		4/11	TCC-002	tc2		*0.50*
					tiago bauer Total	**0.75**
					Grand Total	***0.75***

Figura 16 - Relatório de atividades por projeto e usuário – ClickTime
Fonte: http://www.clicktime.com

5.4 CONSIDERAÇÕES FINAIS

Com está pesquisa foi observado que existem várias ferramentas que gerenciam tempo gasto em tarefas do projeto, mas muitas das ferramentas encontradas não controlam o tempo a partir de ações de iniciar e parara a atividade, sendo que para este controle deve se informar um tempo estimado de realização da tarefa. Portanto, não é possível obter dados como a produtividade de cada membro da equipe, ou o tempo gasto com atividades como reunião,

retrabalho, suportes, entre outras atividade que ocorrem durante o dia-a-dia dos membros da equipe e que colaboram com o insucesso dos projetos. Também se observou que muitas das ferramentas não são usadas somente em ambientes de desenvolvimentos de *software*, mas sim em todas as atividades que requerem gerenciamento de projetos, atividades e tempo gasto.

6 PROPOSTA DA FERRAMENTA DE CONTROLE E MEDIÇÃO DE PRODUTIVIDADE

Este capítulo tem como objetivo abordar quais são as necessidades e as funcionalidades que a ferramenta proposta deverá atender para o controle e a medição de produtividade da equipe de desenvolvimento de *software*. Para isto, será utilizado o embasamento teórico descrito nos capítulos anteriores para justificar as necessidades e funcionalidades.

6.1 NECESSIDADES

Como foi analisado no capítulo 1 deste trabalho, uma das necessidades da medição de produtividade é comparar as entradas, ou seja, o trabalho que deve ser realizado, com as saídas, que é representado pelo esforço produzido para a transformação de uma entrada em uma saída. Esta necessidade tem como objetivo demonstrar quanto esforço uma determinada tarefa precisa para a sua conclusão. Para realizar esta medição é necessário que a ferramenta controle o esforço estimado, como visto no capítulo 3, onde foram descritos os métodos de estimativa de *software*, como por exemplo, a contagem de pontos de função e pontos de caso de uso. Estes métodos serão descritos no texto como contagem de pontos ou somente pontos para facilitar o entendimento.

Outra necessidade encontrada foi de medir a produtividade de cada membro da equipe, através de suas atividades, mas tendo em vista que a produtividade de uma pessoa deve ser comparada com a qualidade do seu serviço. No entanto, para isto deve-se ter como identificar quais os fatores que interferem na produtividade das pessoas e da equipe, como já foi abordado no capítulo 1. Mas também como foi visto deve-se ter como avaliar qual o impacto da tecnologia utilizada sobre a produtividade. Com a utilização destas informações é possível monitorar a necessidade de capacitação da equipe e também ajudará a identificar se algum membro é mais produtivo em uma determinada tecnologia do que em outra. Permitindo-se que em projetos futuros seja possível alocar as pessoas de acordo com a sua produtividade em cada tecnologia.

A partir destas necessidades é possível identificar os indicadores que serão utilizados na medição da produtividade. No capítulo 2 foram abordadas as métricas de *software*, onde foi

analisado que a utilização de métricas é muito importante para melhorar os processos, a qualidade e os custos do projeto, obtendo assim um cronograma mais adequado a equipe e ao tipo de projeto, permitindo a realização de análises sobre quais membros da equipe são mais adequados a determinadas atividades. No entanto, para estas análises será necessária a utilização das métricas já descritas no capítulo 2, como as métricas orientadas a pessoa, para a identificação do desempenho individual de cada membro da equipe, as métricas orientadas a função, devido a utilização pelas empresas de várias linguagens de desenvolvimento nos projetos, esta métrica indicará a quantidade de pontos que cada membro da equipe produz em uma hora. Também com ela pode-se obter a produção de pontos pela equipe, qual é a média de produção diária, entre outras informações que serão melhoradas a cada projeto.

Com a utilização da estimativa de *software* proposta anteriormente, pode-se vinculá-las ao capítulo 4 onde foram abordados dois métodos de controle de produtividade, a partir dos modelos de controle de produtividade propostos no PSP e no TSP. Onde o PSP é o modelo que cada membro da equipe deve seguir, pois ele define que é preciso que cada uma conheça suas fraquezas e habilidades, a partir da especificação de todas as atividades que são realizadas durante o dia, incluindo as interrupções, como auxiliar uma colega, atender ao telefone, entre outras interrupções que a ferramenta deve permitir a identificação.

Ainda com o PSP e TSP deve ser possível registrar a quantidade de erros encontrados no sistema, isto está relacionada à necessidade de controlar estes erros, para isto deve ser possível identificar qual o tipo de erro, conforme descrito na tabela 13 do capítulo 4, e ainda em qual fase foi identificado o erro. A partir desta informação pode-se gerar um novo identificador que poderá ser a quantidade e o tempo gasto com retrabalhos por tarefa ou por membro ou até mesmo por equipe. Também pode se identificar a quantidade de ciclos e qual a quantidade de erros encontrados em cada ciclo até a conclusão da tarefa.

As informações sobre os tipos de erros e as fases de identificação do erro são importantes, pois com elas é possível melhorar o processo ou o projeto onde estão ocorrendo os erros, também é possível identificar onde estão ocorrendo os gargalos que estão atrasando um determinado processo, e até mesmo identificar qual o custo de um erro para o projeto. Para isto a ferramenta também deve ser capaz de controlar o custo da hora de um membro da equipe para que seja possível fazer a comparação mencionada anteriormente.

Outra necessidade observada a partir da análise realizada no capítulo 5, foi a forma de lançamento das horas, que deve ser de forma fácil, ou seja, não sendo burocrática e também não sendo de difícil utilização. Também analisando as ferramentas foi possível observar que elas não controlam a produtividade de cada membro sobre pontos, apenas controlam tempo

estimado e realizado e a produtividade só é possível de se avaliar através dos relatórios. No entanto identificou-se a necessidades de calcular a produtividade a partir da alocação do usuário para a tarefa, já que a produtividade pode variar de pessoa para pessoa. Continuando a análise sobre o capítulo 5, observou que para o lançamento de horas é necessário ter uma tarefa alocada e com isso não é possível lançar, como por exemplo, horas desperdiçadas com suporte telefônico, atendimento a colegas de trabalho, entre outras interrupções já identificadas.

6.2 FUNCIONALIDADES

A partir das necessidades descritas anteriormente pode-se definir quais são as principais funcionalidades do sistema de controle e medição de desenvolvimento de *software*. Estas funcionalidades são:

- O controle do tempo deve ser realizado durante todo o tempo que o membro do projeto estiver trabalhando, para que se possam identificar outras tarefas que interferem no seu trabalho;
- Definir qual o tipo de atividade está sendo realizada durante o expediente e que não esta vinculada a uma tarefa definida para o projeto;
- Permitir iniciar uma atividade sem ter uma tarefa já definida;
- Permitir definir qual o tipo de tecnologia linguagem de programação esta sendo utilizada no desenvolvimento;
- Poder definir o número de pontos da tarefa;
- Permitir encaminhar uma tarefa para outro membro da equipe;
- Permitir reabrir uma tarefa após a sua conclusão;
- Permitir a visualização do status das tarefas *real-time*;
- Permitir o monitoramento das atividades *real-time*;
- O tempo estimado da tarefa deve ser calculado utilizando a quantidade de pontos da tarefa e a produtividade do membro da equipe que foi alocado para a sua realização.

Com a identificação das principais funcionalidades do sistema de controle e medição de produtividade no desenvolvimento de *software*, é possível identificar alguns relatórios que o sistema deverá fornecer para a tomada de decisões do projeto.

- Emitir relatório de comparação entre esforço estimado e esforço realizado;
- Emitir relatório de realização de pontos por membro da equipe;
- Emitir relatório de comparação de pontos por hora;
- Emitir relatório de quantidades de erro:
    - Por membro da equipe;
    - Por projeto;
    - Por equipe.
- Emitir relatório de custo por erro encontrado e solucionado;
- Emitir relatório de custo total do projeto;
- Emitir relatório de atividade fora do projeto que consomem o tempo do projeto;
    - Por Membro;
    - Por equipe.

7 PROJETO DO *SOFTWARE*

Este capítulo tem como objetivo descrever a análise da ferramenta proposta para medição e controle da produtividade da equipe de desenvolvimento de *software*. Esta descrição será feita através da apresentação do modelo Entidade-Relacional (ER) e dos casos de usos. Também será apresentado o diagrama de atividades do sistema, onde será melhor visualizado o fluxo do sistema.

7.1 MODELO ENTIDADE-RELACIONAL (ER)

O modelo de entidade-relacional ou modelo ER como é mais conhecido, tem como objetivo de representar o modelo conceitual de negocio. Na construção do modelo ER procurou-se analisar as informações importantes para o gerenciamento e controle da produtividade. Na figura 17 é apresentado o modelo ER da ferramenta proposta para controle e gerenciamento da produtividade.

A partir do modelo ER é possível realizar uma análise sobre as tabela que foram apresentadas. Começando pela tabela Tarefa que irá conter os registros das tarefas lançadas para o projeto. Já a tabela TarefaUsuario, é responsável por armazenar as informações do usuário alocado para o desenvolvimento da tarefa e qual é o estado atual da tarefa. Outra tabela que requer uma observação é a tabela TarefaErro que irá conter os registros dos erros encontrado durante a execução dos testes da tarefa.

As atividades desenvolvimentos pelos membros da equipe estarão armazenadas na tabela Atividade onde estão as informações de data e hora de início e fim da atividade. Esta tabela esta relacionada com as tabelas de TarefaAtividade e TarefaAtividadeErro que armazenam as atividades correspondentes ao desenvolvimento da tarefa ou dos erros.

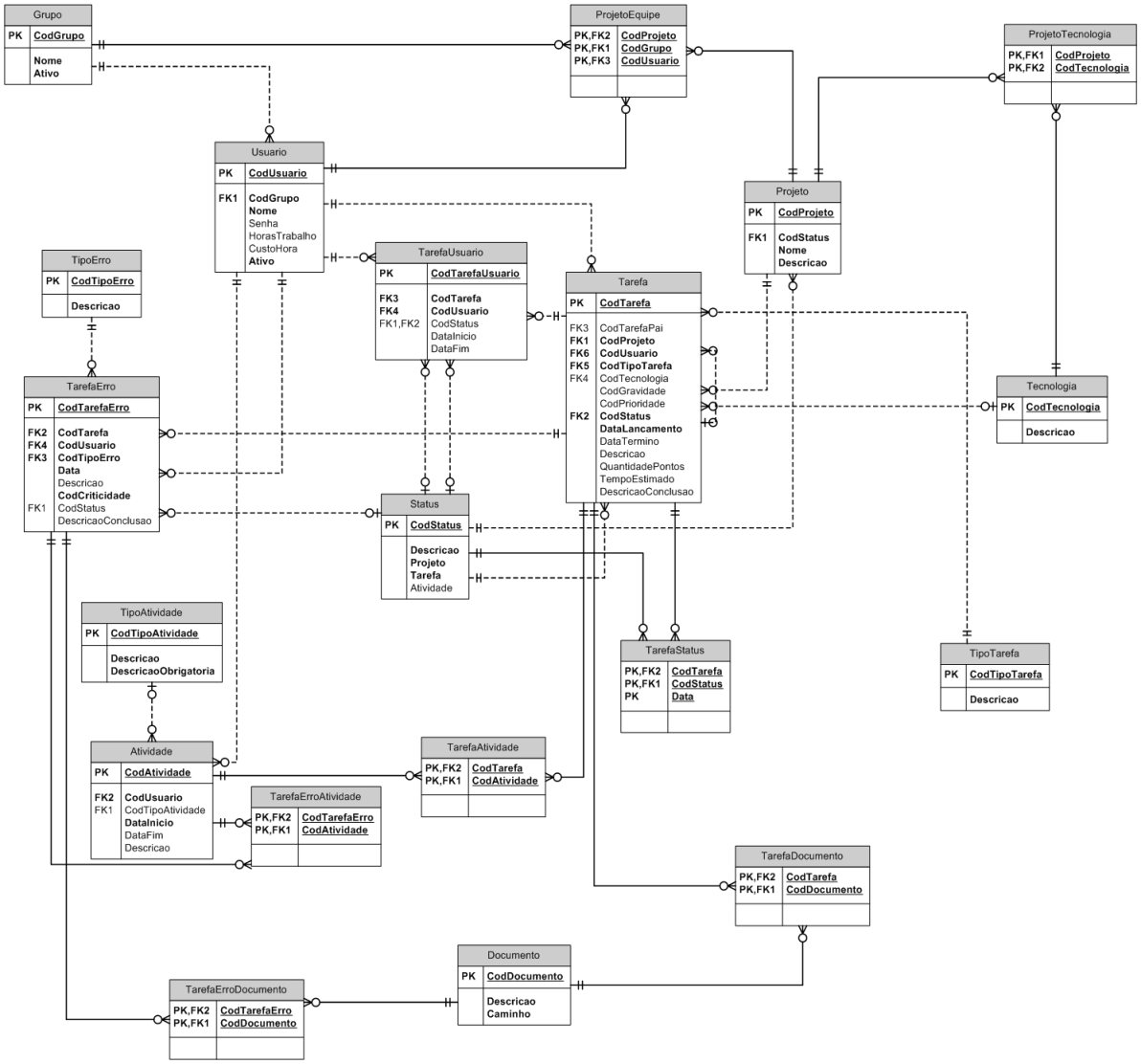


Figura 17 - Modelo ER

7.2 CASOS DE USO

Segundo Fowler (2005), os casos de uso servem para descrever as interações típicas entre os usuários de um sistema e o próprio sistema, fornecendo informações do fluxo de como sistema funciona. Neste trabalho será utilizado o modelo descritivo dos casos de uso, que foi adaptado de Bezerra (2002). Onde será definido o nome, finalidade, atores, mensagens do sistema, regras de negócio (RN), regras de interface (RI), fluxos principais (FP), fluxos alternativos (FA) e os fluxos de exceção (FE). Também serão considerados as informações de definições de campo e seu comportamento e o desenho da tela, para facilitar o desenvolvimento do sistema e a qualidade do produto. A seguir serão apresentados os casos

de uso do sistema que estão representados na figura 18, que apresenta o diagrama de caso de uso. Neste tópico serão apresentados os casos de uso mais importantes para o entendimento do sistema, portanto, não serão apresentados os casos de usos de cadastros básicos da ferramenta.

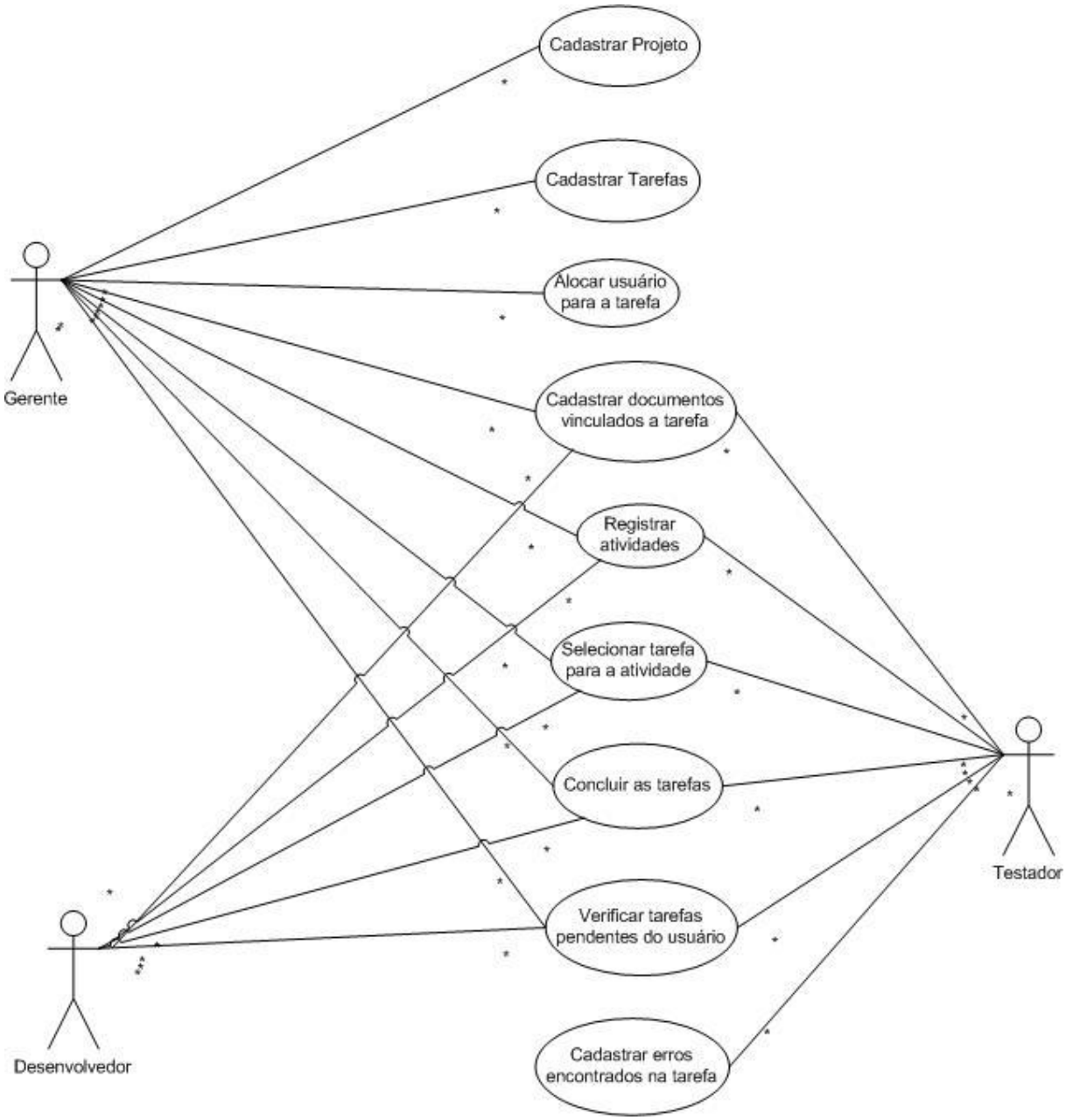


Figura 18 - Diagrama de caso de uso

7.2.1 Cadastrar projeto

Nome do caso de uso: Cadastrar projeto

Finalidade: Cadastrar os projetos que a empresas/setor irá desenvolver.

Atores: Gerentes.

Pré-Condições: Usuário deve ter permissão de gerente para realizar o processo. O campo de tecnologia já deve estar cadastrado e o status já vem definido pelo desenvolvedor.

Interface:

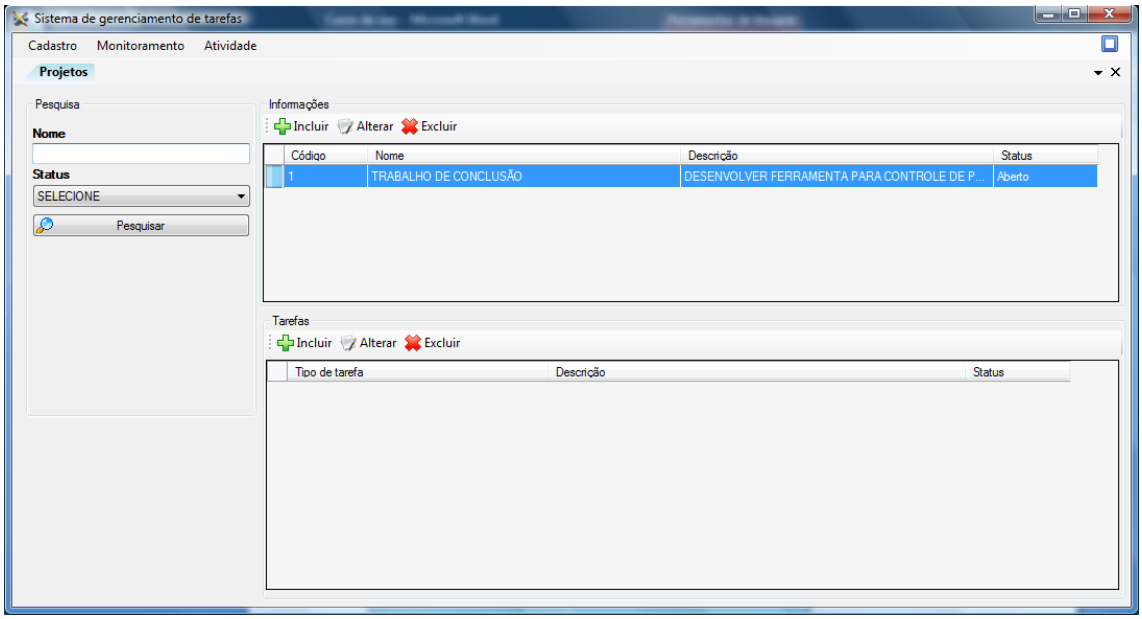


Figura 19 – Interface para consulta dos projetos.

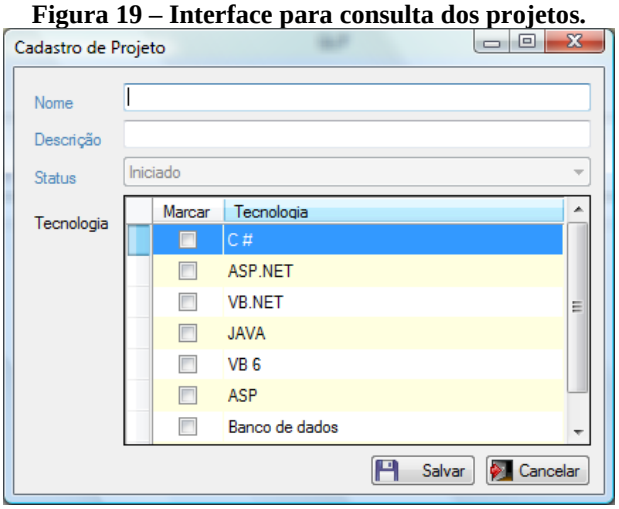


Figura 20 - Interface para cadastro do projeto.

Fluxos principais (FP)

a) Incluir

Ator	**Sistema**
1) Aciona a opção Cadastro, Projeto;	2) Exibe a tela com a lista de projetos já carregada;
3) Aciona a opção Incluir na lista de projetos;	4) Exibe a tela conforme RI1;
5) Informa os dados desejados e aciona a opção Salvar;	6) Valida os dados conforme RN1 e RN2 e exibe a mensagem: "Salvo com sucesso";

b) Alterar

Ator	**Sistema**
1) Aciona a opção Cadastro, Projeto;	2) Exibe a tela com a lista de projetos já carregada;
3) Seleciona na lista de projetos o registro desejado e aciona a opção Alterar na lista de projetos;	4) Exibe a tela com os campos preenchidos e prontos para a alteração;
5) Altera os dados desejados e aciona a opção Salvar;	6) Valida os dados conforme RN1 e RN2 e exibe a mensagem: "Salvo com sucesso";

c) Excluir

Ator	**Sistema**
1) Aciona a opção Cadastro, Projeto;	2) Exibe a tela com a lista de projetos já carregada;
3) Seleciona na lista de projetos o registro desejado e aciona a opção Excluir na lista de projetos;	4) Valida os dados conforme RN3 e exibe a mensagem: "Registro excluído com sucesso";

Regras de negócio (RN)

RN1: Valida o preenchimento dos campos obrigatórios (Em azul). Caso algum campo não esteja preenchido exibe a mensagem: "Verifique o preenchimento dos campos obrigatórios.".

RN2: Não é possível cadastrar o mesmo nome já existente. Caso ocorra, exibe a mensagem "Não é possível inserir registro duplicado.".

RN3: Permitir a exclusão do projeto quando não possuir dependência. Se houver dependência apresentar mensagem "Verifique as dependências.". As dependências serão verificadas através da interface cadastro de tarefas: {Cadastros, Tarefas}.

Regras de interface (RI)

RI1: Exibe a tela conforme a tabela 17 e pronta para a inserção:

Tabela 17 - Estados dos campos da tela de cadastro de projetos.

Campo	Estado	Obrigatório
Nome	Em branco	Sim
Descrição	Em Branco	Sim
Status	Iniciado	Sim
Tecnologia	Lista todas as tecnologias cadastradas e nenhuma marcada	Não

7.2.2 Cadastrar tarefas

Nome do caso de uso: Cadastrar tarefas.

Finalidade: Cadastrar as tarefas do projeto**.**

Atores: Gerente, Analista.

Pré-Condições: Usuário deve ter permissão para realizar o processo. Os campos de tecnologia, projeto, tipos de tarefas já devem estar cadastrado e o status já vem definido pelo desenvolvedor.

Interface:

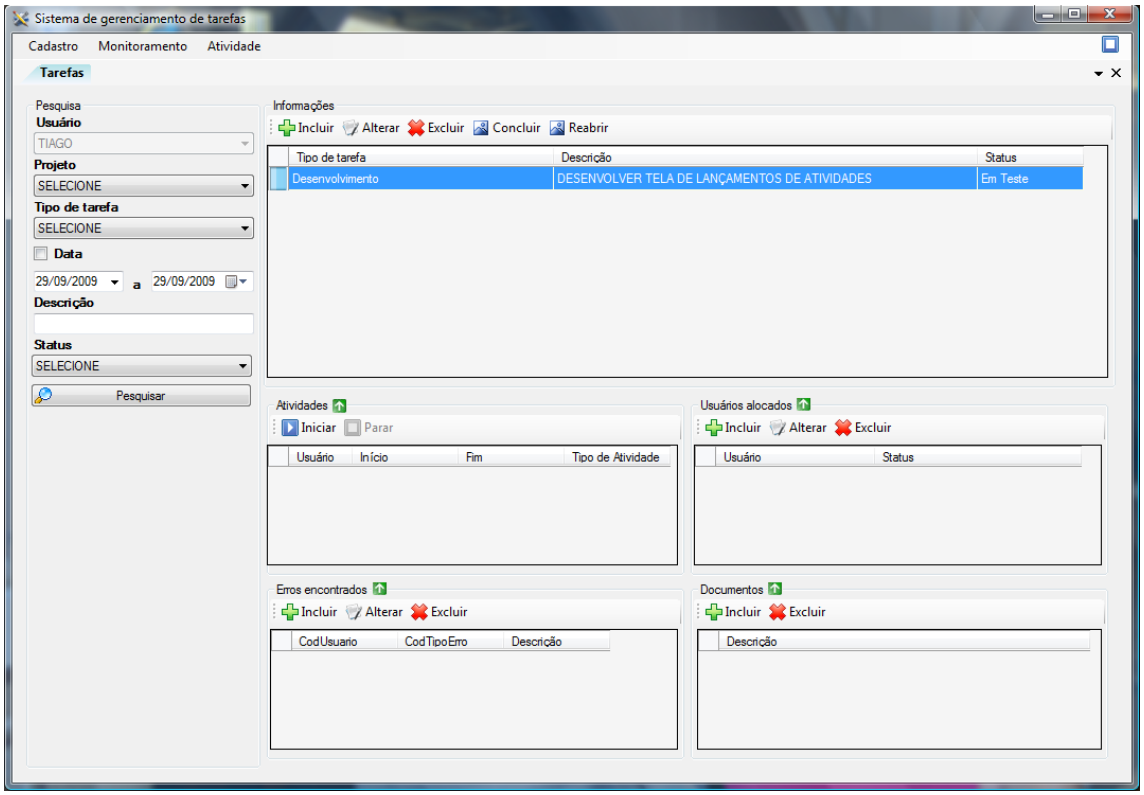


Figura 21 - Interface para consulta de tarefas.

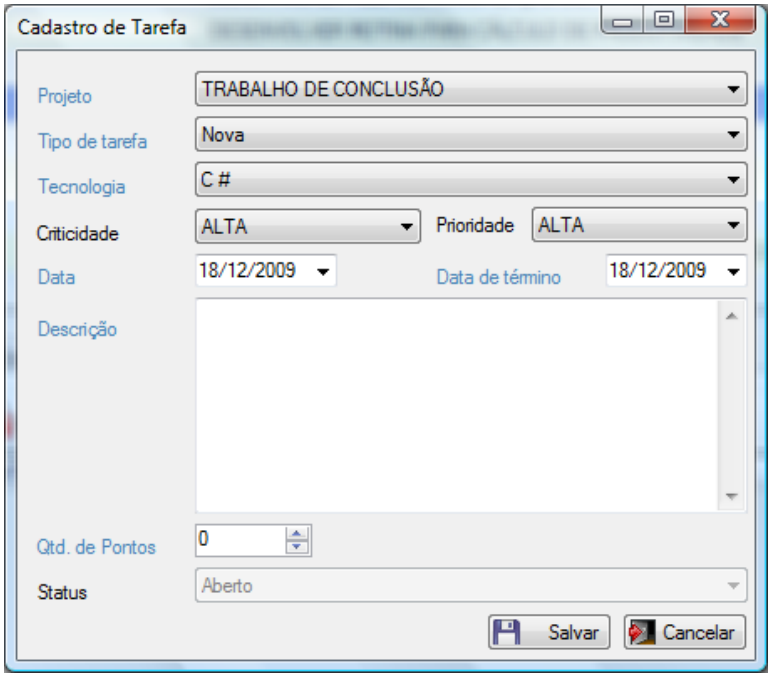


Figura 22 – Interface para cadastro de tarefas.

Fluxos principais (FP)

a) Incluir

Ator	**Sistema**
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Aciona a opção Incluir na lista de tarefas;	4) Exibe a tela conforme RI1;
5) Informa os dados desejados e aciona a opção Salvar;	6) Valida os dados conforme RN1 e exibe a mensagem: "Salvo com sucesso";

b) Alterar

Ator	**Sistema**
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado e aciona a opção Alterar na lista de tarefas;	4) Exibe a tela com os campos em preenchidos e prontos para a alteração;
5) Altera os dados desejados e aciona a opção Salvar;	6) Exibe a mensagem: "Salvo com sucesso";

c) Excluir

Ator	**Sistema**
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado e aciona a opção Excluir na lista de tarefas	4) Valida os dados conforme RN2 e exibe a mensagem "Registro excluído com sucesso";

Regras de negócio (RN)

RI1: Valida o preenchimento dos campos obrigatórios (Em azul). Caso algum campo não esteja preenchido exibe a mensagem: “Verifique o preenchimento dos campos obrigatórios.”.

RN2: Permitir a exclusão do projeto quando não possuir dependência. Se houver dependência apresentar mensagem “Verifique as dependências.”. As dependências serão verificadas através da interface de cadastro de tarefas: {Cadastros, Tarefas}.

Regras de interface (RI)

RI1: Exibe a tela conforme a tabela 18 e pronta para a inserção:

Tabela 18 - Estados dos campos da tela de cadastro de tarefas.

Campo	Estado	Obrigatório	Bloqueia inclusão
Projeto	Selecione	Sim	Não
Tipo de tarefa	Selecione	Sim	Não
Tecnologia	Selecione	Sim	Não
Criticidade	Selecione	Não	Não
Prioridade	Selecione	Não	Não
Data	Data atual	Sim	Não
Data de término	Data atual	Sim	Não
Descrição	Em Branco	Sim	Não
Qtd. de pontos	0	Sim	Não
Status	Aberto	Não	Sim

7.2.3 Alocar usuário para a tarefa.

Nome do caso de uso: Alocar usuário para a tarefa.

Finalidade: Cadastrar os usuários alocados para tarefa**.**

Atores: Gerentes.

Pré-Condições: Usuário deve ter permissão de gerente para realizar o processo. O campo status já vem definido pelo desenvolvedor.

Interface:

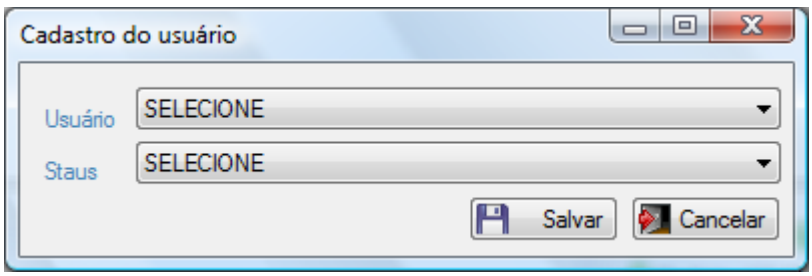


Figura 23 - Interface para alocação de usuário.

Fluxos principais (FP)

a) Incluir

Ator	**Sistema**
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado;	4) Carrega a lista de usuários alocados;
5) Aciona a opção Incluir na lista de usuários alocados;	6) Exibe a tela conforme RI1;
7) Informa os dados desejados e aciona a opção Salvar;	8) Valida os dados conforme RN1 e exibe a mensagem: "Salvo com sucesso";

b) Alterar

Ator	**Sistema**
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado;	4) Carrega a lista de usuários alocados;
5) Seleciona na lista de usuários alocados o registro desejado e aciona a opção Alterar na lista de usuários alocados;	6) Exibe a tela com os campos em preenchidos e prontos para a alteração e conforme RI1;
7) Altera os dados desejados e aciona a opção Salvar;	8) Exibe a mensagem: "Salvo com sucesso";

c) Excluir

Ator	**Sistema**
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado;	4) Carrega a lista de usuários alocados;
5) Seleciona na lista de usuários alocados o registro desejado e aciona a opção Excluir na lista de usuários alocados;	6) Valida os dados conforme RN2 e exibe a mensagem "Registro excluído com sucesso";

Regras de negócio (RN)

RI1: Valida o preenchimento dos campos obrigatórios (Em azul). Caso algum campo não esteja preenchido exibe a mensagem: "Verifique o preenchimento dos campos obrigatórios.".

RN2: Permitir a exclusão do projeto quando não possuir dependência. Se houver dependência apresentar mensagem "Verifique as dependências.". As dependências serão verificadas através da interface de cadastro de tarefas: {Cadastros, Tarefas}.

Regras de interface (RI)

RI1: Exibe a tela conforme a tabela 19 e pronta para a inserção:

Tabela 19 - Estados dos campos da tela de alocação de usuário

Campo	Estado	Obrigatório	Bloqueio alteração
Usuário	Selecione	Sim	Sim
Status	Selecione	Sim	Não

7.2.4 Cadastrar documentos vinculados a tarefa.

Nome do caso de uso: Cadastrar documentos vinculados a tarefa.

Finalidade: Cadastrar os documentos vinculados a tarefa**.**

Atores: Gerente, Analista, desenvolvedores e Testadores.

Pré-Condições: Usuário deve ter permissão para realizar o processo.

Interface:

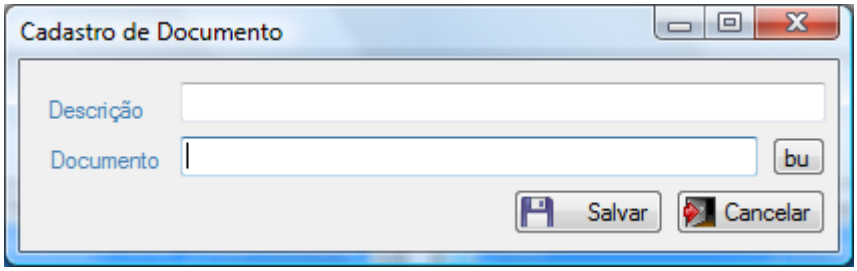


Figura 24 – Interface para cadastro de documentos vinculados a tarefa.

Fluxos principais (FP)

a) Incluir

Ator	Sistema
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado;	4) Carrega a lista de documentos;
5) Aciona a opção Incluir na lista de documentos;	6) Exibe a tela conforme RI1;
7) Informa os dados desejados e aciona a opção Salvar;	8) Exibe a mensagem: "Salvo com sucesso";

b) Alterar

Ator	Sistema
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado;	4) Carrega a lista de documentos;

5) Seleciona na lista de documentos o registro desejado e aciona a opção Alterar na lista de documentos;	6) Exibe a tela com os campos em preenchidos e prontos para a alteração;
7) Altera os dados desejados e aciona a opção Salvar;	8) Exibe a mensagem: "Salvo com sucesso";

c) Excluir

Ator	**Sistema**
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado;	4) Carrega a lista de documentos;
5) Seleciona na lista de documentos o registro desejado e aciona a opção Excluir na lista de documentos;	6) Exibe a mensagem "Registro excluído com sucesso";

Regras de negócio (RN)

RI1: Valida o preenchimento dos campos obrigatórios (Em azul). Caso algum campo não esteja preenchido exibe a mensagem: "Verifique o preenchimento dos campos obrigatórios.".

Regras de interface (RI)

RI1: Exibe a tela conforme a tabela 20 e pronta para a inserção:

Tabela 20 - Estados dos campos da tela de cadastro de documentos.

Campo	Estado	Obrigatório
Descrição	Em branco	Sim
Caminho	Em branco	Sim

7.2.5 Cadastrar erros encontrados na tarefa.

Nome do caso de uso: Cadastrado erros encontrados na tarefa.

Finalidade: Cadastrar os erros encontrados na tarefa**.**

Atores: Testadores.

Pré-Condições: Usuário deve ter permissão para realizar o processo. O campo tipo de erro já deve estar cadastro e os campos criticidade e status já vêm definidos pelo desenvolvedor.

Interface:

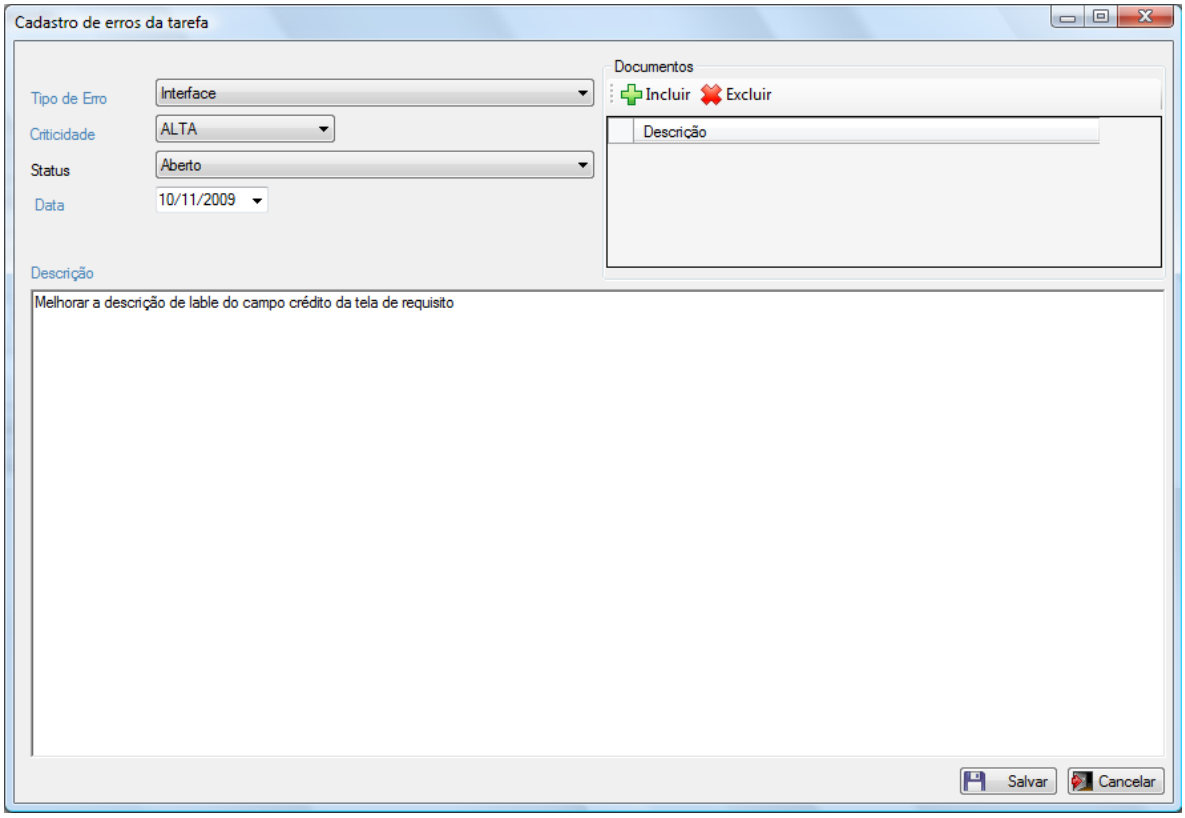


Figura 25 - Interface para cadastro de erros da tarefa.

Fluxos principais (FP)

a) Incluir

Ator	**Sistema**
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado;	4) Carrega a lista de erros;
5) Aciona a opção Incluir na lista de erros;	6) Exibe a tela conforme RI1;
7) Informa os dados desejados e aciona a opção Salvar;	8) Valida os dados conforme RN1 e exibe a mensagem: “Salvo com sucesso”;

b) Alterar

Ator	Sistema
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado;	4) Carrega a lista de erros;
5) Seleciona na lista de erros o registro desejado e aciona a opção Alterar na lista de erros;	6) Exibe a tela com os campos em preenchidos e prontos para a alteração;
7) Altera os dados desejados e aciona a opção Salvar;	8) Valida os dados conforme RN1 e exibe a mensagem: “Salvo com sucesso”;

c) Excluir

Ator	Sistema
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado;	4) Carrega a lista de documentos;
5) Seleciona na lista de erros o registro desejado e aciona a opção Excluir na lista de erros;	6) Valida os dados conforme RN2 e exibe a mensagem "Registro excluído com sucesso";

Regras de negócio (RN)

RN1: Valida o preenchimento dos campos obrigatórios (Em azul). Caso algum campo não esteja preenchido exibe a mensagem: "Verifique o preenchimento dos campos obrigatórios.".

RN2: Permitir a exclusão doe erros quando não possuir dependência. Se houver dependência apresentar mensagem "Verifique as dependências.". As dependências serão verificadas através da interface de cadastro de tarefas:

- {Cadastros, Tarefas, lista de erros, lista de documentos}.
- {Cadastros, Tarefas, lista de atividades}.

Regras de interface (RI)

RI1: Exibe a tela conforme a tabela 21 e pronta para a inserção:

Tabela 21 - Estados dos campos da tela de cadastro de erros.

Campo	Estado	Obrigatório	Bloqueia inclusão
Tipo de erro	Selecione	Sim	Não
Criticidade	Selecione	Sim	Não
Data	Atual	Sim	Não
Descrição	Em branco	Sim	Não
Status	Aberto	Não	Sim

7.2.6 Registrar atividades.

Nome do caso de uso: Registrar atividades.

Finalidade: Cadastrar as atividades que estão sendo realizadas**.**

Atores: Gerente, Analista, desenvolvedores e Testadores.

Pré-Condições: Usuário deve ter permissão para realizar o processo.

Interface:

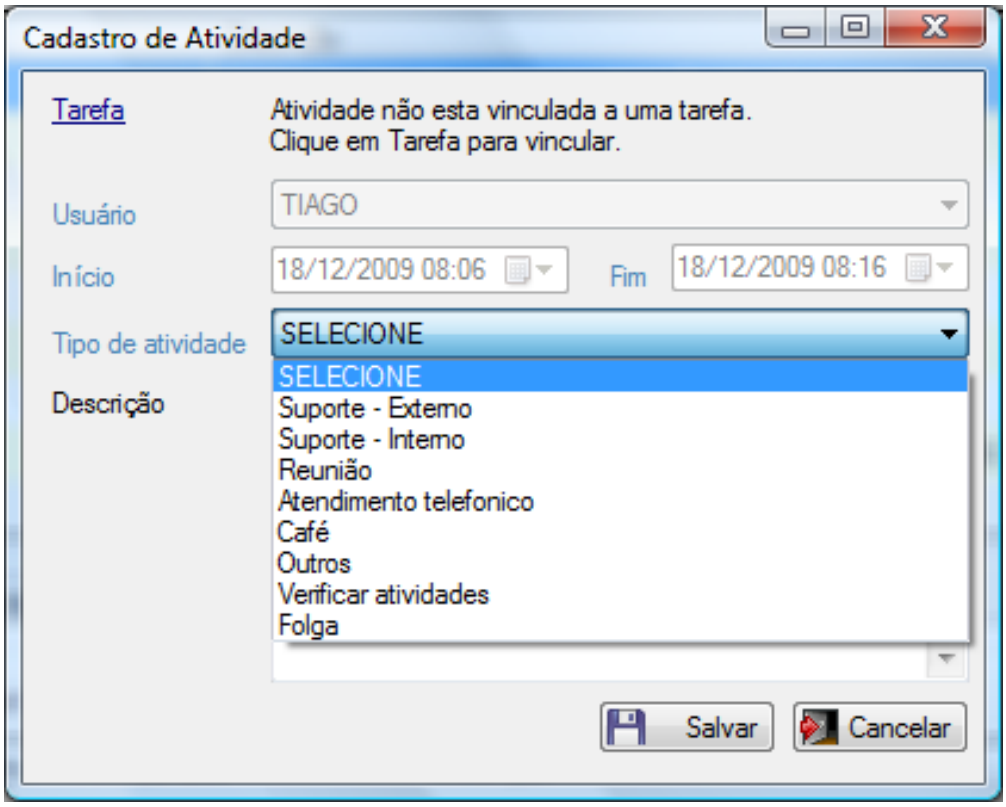


Figura 26 – Interface para cadastro de atividades.

Fluxos principais (FP)

a) Iniciar

Ator	**Sistema**
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado;	4) Carrega a lista de atividades;
5) Aciona a opção iniciar na lista de atividades;	6) Valida os dados conforme RN1 e inicia a atividade;

Parar

Ator	**Sistema**
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado;	4) Carrega a lista de erros;
5) Seleciona na lista de atividades o registro desejado e aciona a opção parar na lista de atividades;	6) Exibe a tela com os campos em preenchidos e prontos para a alteração;
7) Altera os dados desejados e aciona a opção Salvar;	8) Valida os dados conforme RN2;

Fluxos alternativos (FA)

- Vincular uma tarefa a atividade. Processo descrito no caso de uso: Selecionar tarefa para a atividade - 7.2.7.
- Concluir a atividade. Processo descrito neste caso de uso (2.7.6), seguindo o fluxo "b".

Regras de negócio (RN)

RN1: Valida se não existe nenhuma atividade iniciada para o usuário. Caso exista, exibir a interface de cadastro de atividade para a realização da conclusão da atividade e iniciar a nova atividade.

RN2: Valida o preenchimento dos campos obrigatórios (Em azul). Caso algum campo não esteja preenchido exibe a mensagem: "Verifique o preenchimento dos campos obrigatórios.".

Regras de interface (RI)

RI1: Exibe a tela conforme a tabela 22 e pronta para a inserção:

Tabela 22 - Estado dos campos da tela de cadastro de atividades.

Campo	Estado	Obrigatório
Fim	Atual	Sim
Tipo de atividade	Selecione	Sim
Descrição	Em branco	Somente para alguns tipos de atividades

RI2: Exibe a tela com os campos usuário e início bloqueados para alteração.

7.2.7 Selecionar tarefa para a atividade.

Nome do caso de uso: Selecionar tarefa para a atividade.

Finalidade: Selecionar uma tarefa para uma atividade que foi inicializada sem vínculo com uma tarefa**.**

Atores: Gerente, analista, desenvolvedores e testadores.

Pré-Condições: Usuário deve ter permissão para realizar o processo.

Interface:

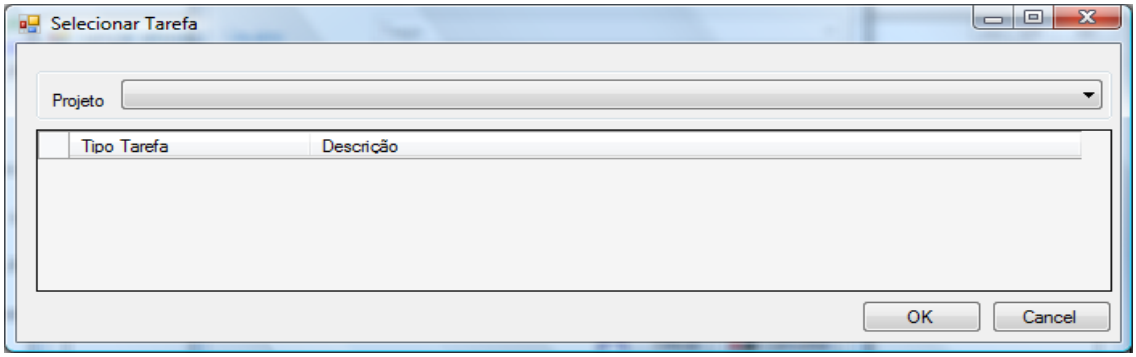


Figura 27 - Interface para selecionar a tarefa.

Fluxos principais (FP)

a) Selecionar

Ator	Sistema
1) Clica sobre o sublinhado da tela de atividades;	2) Exibe a tela com o campo projeto, carregado com os projetos já cadastrados e a lista de tarefas pendentes para o projeto;
3) Seleciona na lista de tarefas o registro desejado e aciona o botão "OK";	4) Define a tarefa selecionada para a atividade;

7.2.8 Concluir as tarefas

Nome do caso de uso: Concluir as tarefas.

Finalidade: Concluir as tarefas pendentes, permitindo o encaminhamento da tarefa para outra equipe ou pessoa.

Atores: Gerente, analista, desenvolvedores e testadores.

Pré-Condições: Usuário deve ter permissão para realizar o processo.

Interface:

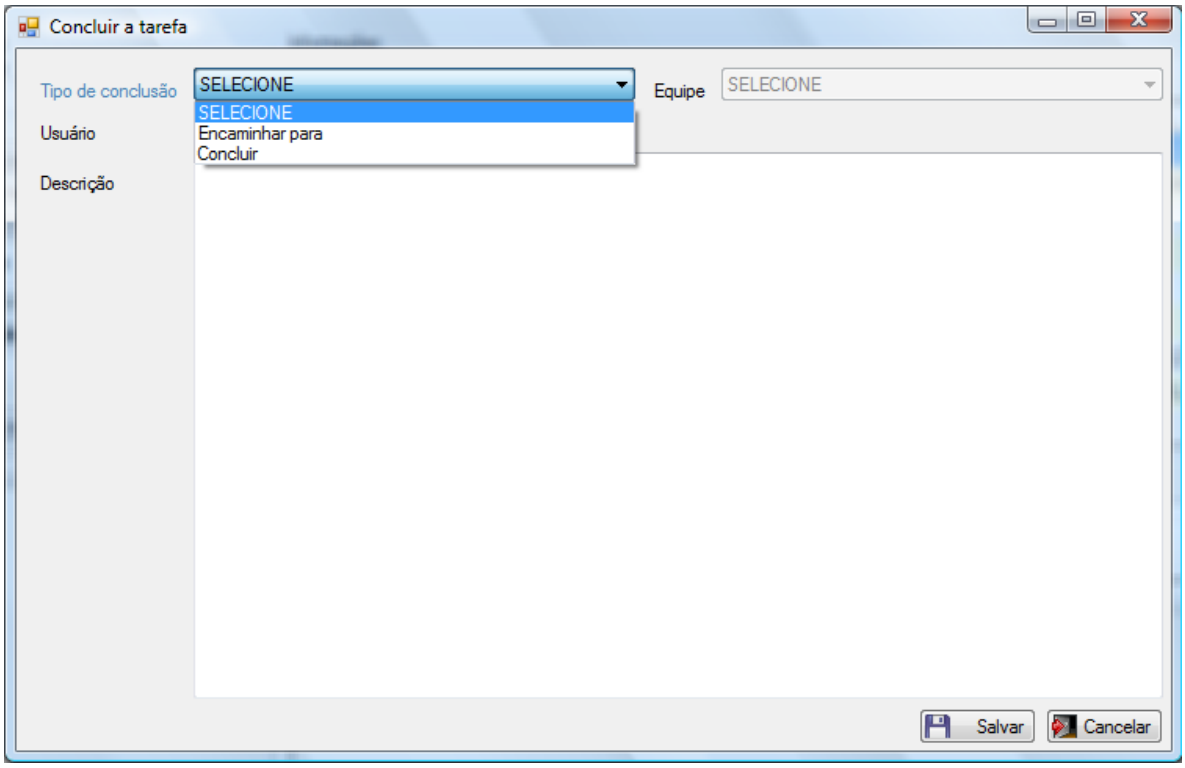


Figura 28 - Interface para conclusão de tarefas.

Fluxos principais (FP)

a) Concluir uma tarefa pendente

Ator	**Sistema**
1) Aciona a opção Cadastro, Tarefa;	2) Exibe a tela com a lista de tarefas já carregada;
3) Seleciona na lista de tarefas o registro desejado e clique sobre o botão "concluir";	4) Abre a tela de conclusão de tarefas;
5) Informa os dados desejados e clique sobre o botão "Salvar";	6) Valida os dados conforme RN1 e fecha tela;

Fluxos alternativos (FA)

- Cadastro de atividade. Processo descrito no caso de uso 2.7.

Regras de negócio (RN)

RN1 – Valida se existe alguma atividade já iniciada, caso existe abre a tela para a conclusão da atividade.

7.2.9 Verificar tarefas pendentes do usuário

Nome do caso de uso: Verificar tarefas pendentes do usuário.

Finalidade: Verificar as tarefas pendentes para o usuário, facilitando o acesso e permitindo o lançamento de atividades através desta interface.

Atores: Gerente, analista, desenvolvedores e testadores.

Pré-Condições: Usuário deve ter permissão para realizar o processo.

Interface:

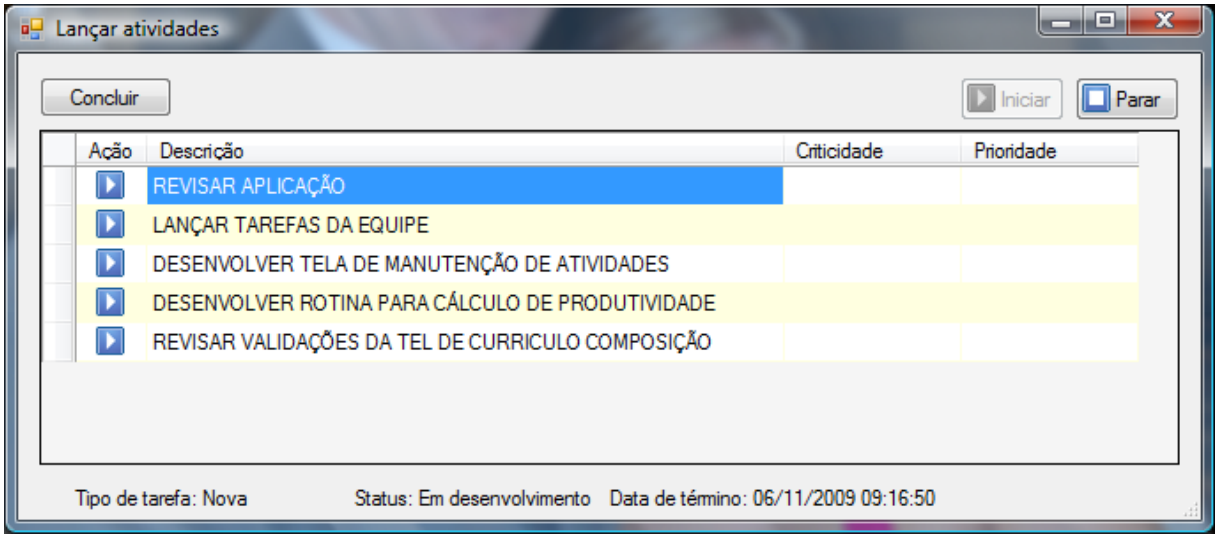


Figura 29 – Interface para verificar as tarefas pendentes

Fluxos principais (FP)

a) Iniciar uma tarefa através de tela de tarefas pendentes

Ator	**Sistema**
1) Clicar duas vezes sobre o ícone do sistema.	2) Exibe a tela com a lista de tarefas pendentes já carregadas;
3) Seleciona na lista de tarefas o registro desejado e aciona o botão “Iniciar”;	4) Valida RN1 inicia a atividade e fecha a tela;

b) Parar uma tarefa através de tela de tarefas pendentes

Ator	**Sistema**
1) Clicar duas vezes sobre o ícone do sistema.	2) Exibe a tela com a lista de tarefas pendentes já carregadas;
3) Seleciona na lista de tarefas o registro desejado e aciona o botão “Parar”;	4) Abre a tela para conclusão da atividade;

c) Concluir uma tarefa através de tela de tarefas pendentes

Ator	**Sistema**
1) Clicar duas vezes sobre o ícone do sistema.	2) Exibe a tela com a lista de tarefas pendentes já carregadas;
3) Seleciona na lista de tarefas o registro desejado e aciona o botão “Concluir”;	4) Abre a tela para conclusão da tarefa;

Fluxos alternativos (FA)

- Cadastro de atividade. Processo descrito no caso de uso 2.7.2 – Cadastrar as tarefas;
- Conclusão de tarefas. Processo descrito no caso de uso 2.7.8 – Concluir as tarefas;

Regras de negócio (RN)

RN1 – Valida se existe alguma atividade já iniciada, caso existe abre a tela para a conclusão da atividade.

7.3 FLUXOGRAMA

O fluxograma das atividades exibido na figura 29 tem como propósito representar de forma gráfica todo o processo desde o cadastro do projeto e a definição da equipe, passando pelo cadastro de tarefas e atividades até a conclusão da tarefa que se dá a partir da conclusão da tarefa pela equipe de testes.

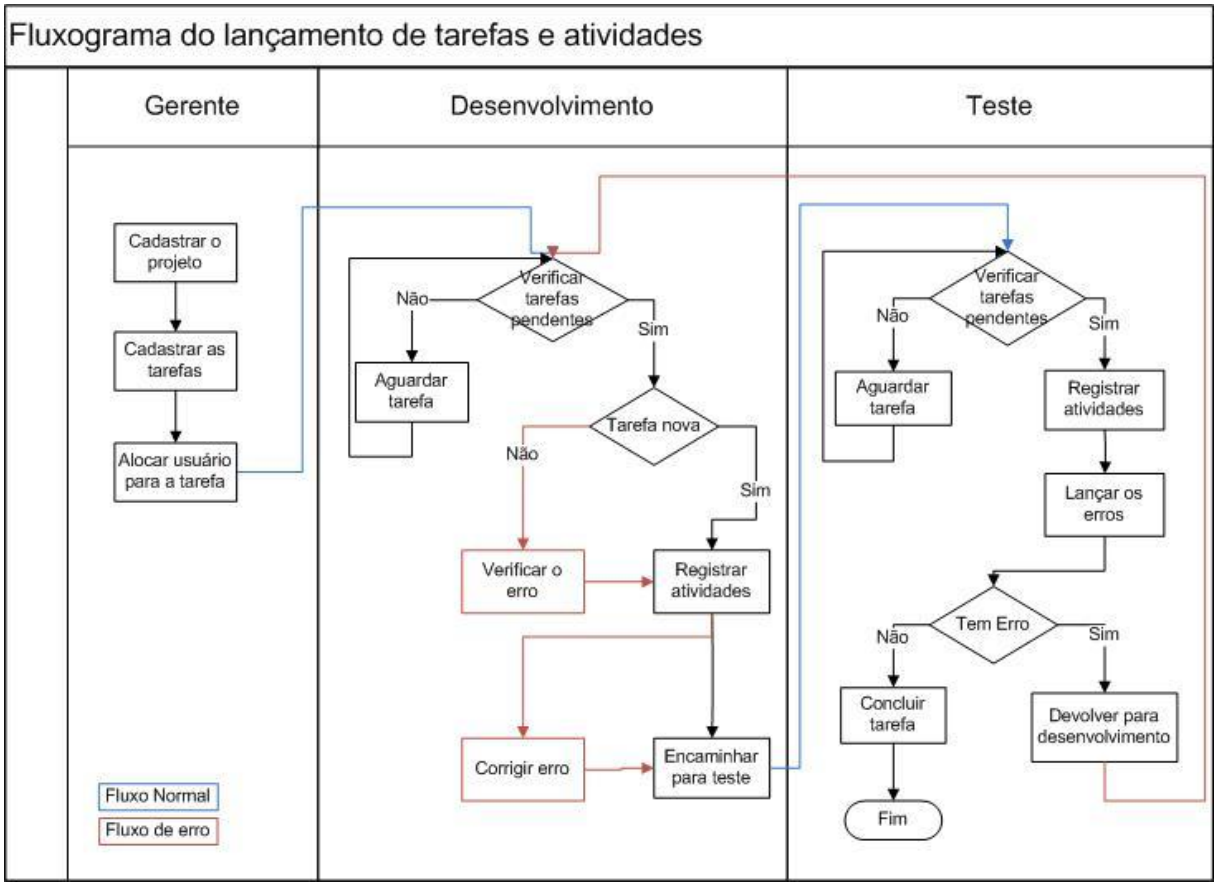


Figura 30 – Fluxograma.

Analisando o fluxograma é possível identificar em qual etapa cada membro da equipe deverá atuar, ou seja, o gerente ou analista é responsável pelo cadastrado do projeto e as tarefas, também é de sua responsabilidade alocar as tarefas para os desenvolvedores. Este três passos foram descritos nos casos de uso 7.2.1, 7.2.2 e 7.2.3 onde é possível analisar as regras e as informações contidas nos cadastros. A verificação das tarefas pode ser analisada através do caso de uso 7.29. Já os registros das atividades da tarefa são observados através do caso de uso 7.2.6. Estes dois passos são comuns as equipes de desenvolvimento e teste. No entanto, para a equipe de teste tem-se o passo de lançar os erros que foi descrito no caso de uso 7.2.5. Caso seja encontrados erros na tarefa, esta deve ser encaminhada para o desenvolvimento realizar as correções, e após as correções realizadas, deverá voltar para os testes. Este ciclo se repetirá até que os testes não encontrem mais erros e a tarefa possa ser concluída. Este processo de conclusão ou encaminhamento das tarefas foi melhor descrito no caso de uso 7.2.8.

7.4 TECNOLOGIA

A escolha da tecnologia que foi utilizada no desenvolvimento da ferramenta se deu através do conhecimento do autor, e também do fato da empresas onde se identificou a necessidade de controle da produtividade já utilizarem está tecnologia. Portanto, a ferramenta foi desenvolvida sobre a plataforma Microsoft, sendo que para a construção do banco de dados foi utilizado o *SQL Server Express 2008*. Já para desenvolver as camadas de negócio e apresentação da ferramenta foi utilizado o *Visual Studio 2008,* sendo que foi utilizado *Windows forms* e *C# (Sharp)* para as suas construções. Já os relatórios foram desenvolvidos com ajuda do pacote do c*ristal reports.*

7.5 RELATÓRIOS

No capítulo 6 foram apontados a necessidades de alguns relatórios para o gerenciamento e controle da produtividade da equipe de desenvolvimento de sistemas.

7.5.1 Relatório de comparação entre esforço estimado e esforço realizado

Este relatório tem como objetivo demonstrar o tempo realizado das tarefas no projeto, este tempo pode ser obtido a partir dos lançamentos de atividades da tarefa, comprando com o tempo estimado, que é obtido a partir do cálculo entre a produtividade do usuário alocado para o desenvolvimento da tarefa e a quantidade de pontos definido no cadastro da tarefa. O relatório pode ser gerado por projeto, equipe e usuário, esta informação pode ser obtida a qualquer momento.

7.5.2 Relatório de realização de pontos por membro da equipe

O relatório de realização de pontos por membro tem como objetivo medir a quantidade de pontos que cada membro da equipe produz por hora. Com este relatório também é possível visualizar a produção de pontos em diferentes linguagens, pois o relatório será agrupado por usuário e tecnologia.

7.5.3 Relatório de produtividade

O relatório de produtividade tem como objetivo apresentar a produtividade individual de cada membro da equipe. Esta informação é importante, pois além de ser a produtividade individual de cada membro, ela também é importante ao se calcular o tempo estimado de cada tarefa.

7.5.4 Relatório de ciclos de uma tarefa entre as equipes

O relatório de ciclos de uma tarefa entre as equipes tem o intuito de mostrar quantos ciclos então ocorrendo entre as equipes, como por exemplo, a quantidade de vezes que a tarefa passou pela equipe de testes e voltou para o desenvolvimento com erros. Esta informação é importante para analisar possíveis gargalos que estejam ocorrendo no decorrer do projeto.

7.5.5 Relatório de quantidades de erro

Este relatório tem o objetivo de analisar a quantidade de erros que estão ocorrendo por tarefa e no projeto. A geração deste relatório pode ser feita para visualizar a quantidade de erros por tarefa do alocada para o desenvolvedor.

7.5.6 Relatório de custo por erro encontrado e solucionado

Com o relatório de custo por erro encontrado e solucionado pode se analisar e mostrar a equipe qual o custo de um erro no sistema, e com isso pode ser melhorado a equipe ou

processo de desenvolvimento de *software*, desde a geração de documentos de requisitos até a entrega do projeto, podendo identificar em qual fase o erro foi encontrado e solucionado.

7.5.7 Relatório de atividade fora do projeto que consomem o tempo do projeto

Este relatório tem o objetivo de apresentar as atividades que estão ocorrendo fora do escopo do projeto. Com esta informação é possível mitigar possível gargalos no projeto, e fornecer ao gerente informações para o gerente tomar ações para melhor a capacidade produtiva da equipe ou melhorar os processos.

7.6 OUTRAS FUNCIONALIDADES

Para facilitar o acesso para os lançamentos das atividades, foi adicionada a barra de tarefas do *Windows* um ícone, conforme apresentado na figura 30. A partir deste ícone é possível iniciar ou parar uma atividade, acessar a tela inicial do sistema, onde é possível acessar aos menus de controle de projeto, tarefas, relatórios, entre outras telas do sistema. Outra funcionalidade acessada a partir deste ícone são as tarefas pendentes. Para isto é necessário clicar duas vezes sobre o ícone para abrir a tela de tarefas pendentes.

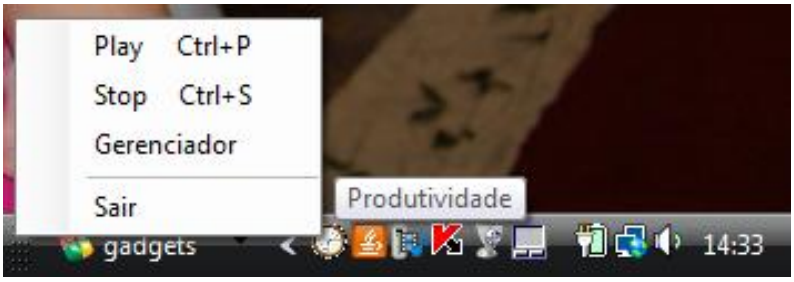


Figura 31 - Atalho para iniciar e parar as atividades

8 ESTUDO DE CASO

Para a validação da ferramenta proposta viu-se a necessidade da realização do estudo de caso em uma empresa ou setor de desenvolvimento de *software*. Portanto, foi escolhido um setor de desenvolvimento de *software*, dentro de uma instituição de ensino de Novo Hamburgo. Este setor é responsável em planejar e executar melhorias, implementações novas ou aquisições de sistemas de informação, encaminhando e esclarecendo qualquer ação que deva ser tomada à cerca de suas necessidades. Estes planejamentos ou execuções sempre são decididos juntamente com coordenadores ou usuários líderes de outras áreas.

Também é de competência do setor atender, mediante análise, demandas de informação, registro e controle das áreas convertendo-as, conforme disponibilidade, em funcionalidades e/ou sistemas de informação. Planejar ações para garantir, na maior parte do tempo, disponibilidade e bom funcionamento de todo e qualquer sistema de informação bem como os respectivos planos de contingência. Identificar oportunidade de melhorias em processos, sendo eles informatizados ou não e encaminhar, juntamente com os respectivos coordenadores de área e / ou usuários líderes, ações para encaminhamento da melhoria.

Para a realização do estudo de caso, foram escolhidas algumas pessoas que são responsáveis por um projeto novo na instituição que atualmente encontra-se em fase de pós-implantação. O projeto começou a ser desenvolvido na metade de 2008, sendo que antes disto foi realizada a análise dos requisitos e a modelagem do sistema, que durou aproximadamente um ano e meio. Após isso foi realizada a estimativa de esforço e tamanho do projeto utilizando-se a análise de pontos de função. A partir da estimativa realizada viu-se a necessidade de dividir o projeto em fases, já que seu desenvolvimento é de aproximadamente quatro anos. Sendo que em cada fase o projeto passa por três ciclos de desenvolvimento, sendo eles: desenvolvimento das regras de negócio, desenvolvimento da interface, que para está fase é utilizando um gerador de interfaces, que foi desenvolvido pelo próprio setor, e a realização de testes do sistema.

No entanto, para a validação da ferramenta, estabeleceu-se uma semana a realização do estudo de caso. A partir disto foram estabelecidas as tarefas que cada membro da equipe teria para desenvolver durante os dias em que seriam coletadas as informações. Como este estudo não tinha como objetivo direcionar as pessoas a realizarem atividades fora do seu cotidiano, foi definido as atividades que estavam vinculadas ao projeto. Entretanto, foi solicitado para que além de lançarem o esforço necessário para a realização da tarefa, fossem informadas

outras atividades que eles teriam que realizar durante a execução da tarefa. Estas atividades foram sendo analisadas durante o trabalho como: suporte interno (auxiliar o colega de trabalho na solução de algum problema), suporte externo (Atender algum usuário fora do setor), atendimento telefônico, folga, reunião e outros (banheiro, café, etc.).

Também foi informada aos membros da equipe a importância de lançarem as atividades de forma correta, pois esta informação é de extrema importância para o cálculo da produtividade de cada membro.

8.1 RESULTADOS

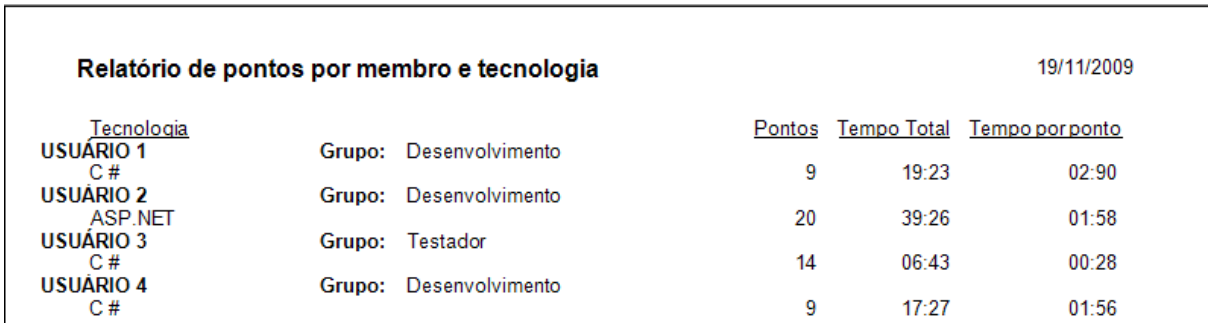
A partir do estudo de caso realizado, foi possível avaliar a utilização da ferramenta e realizar a análise dos dados obtidos. Para a avaliação da utilização da ferramenta, não foi estabelecido nenhum questionário, apenas foram realizadas conversas informais sobre a sua utilização. Com isso, foi possível avaliar a facilidade de lançar novas as atividades, que é realizado através da funcionalidade descrita no capítulo 7 e pode ser observado na figura 30.

Outro ponto que foi destacado foi a forma encontrada para gerenciar as atividades entre as equipes, principalmente desenvolvimento e testes, onde se encontra o maior número de interações, esta funcionalidade foi apresentada nos casos de uso e também no diagrama de atividades do capítulo 7. Também foi solicitado pelos usuários uma tela para a alteração dos registros de atividades, pois muitas vezes ocorrem casos em que foi esquecido de parar uma atividade ou iniciar uma nova atividade e pela ferramenta proposta não tinha como lançar esta alteração.

No entanto, a proposta era uma ferramenta de controle e medição da produtividade no desenvolvimento de *software*, e não uma ferramenta para lançamento de atividades para tarefas. Mas para medir-se a produtividade é necessário conhecer as atividades que fazem parte do dia-a-dia dos membros de uma equipe. Por isto, foi necessário o desenvolvimento de uma ferramenta para a realização deste controle das atividades. Conforme visto no capítulo 5, muitas ferramentas encontradas no mercado controlam o esforço para a realização de uma determinada tarefa, mas não permitem lançar atividades que acabam ocorrendo sem planejamento, como já foi citado anteriormente.

A seguir serão apresentados os relatórios que podem ser geradas a qualquer momento e que trazem informações referentes a equipe, tarefas e usuários. Como o tempo para a realização deste estudo de caso não foi muito extenso, e também não tendo um histórico da produtividade da equipe em outros projetos, as informações obtidas são válidas para o cálculo inicial da produtividade. Para o cálculo da produtividade é feito uma média entre todos os tempos já realizados em atividades que tenham definidas tipos de atividades que controlam a produtividade, como por exemplo, desenvolvimento e a quantidade total de pontos que já foram concluídos em tarefas. Pois somente a partir da conclusão das tarefas é possível avaliar a produtividade individual de cada membro, para que posteriormente ao ser alocado uma nova tarefa para este membro seja possível calcular o tempo estimado utilizando como base a produtividade até aquela data obtida multiplicado pelo número de pontos da tarefa. Para este cálculo sempre será obedecida a produtividade de cada usuário sobre a tecnologia empregada para a tarefa. Outro problema que foi encontrado, é que devido ao lançamento de atividades durante o dia não ser uma cultura das pessoas, muitas vezes acabavam esquecendo-se de registrar suas atividades.

Na figura 31 é apresentado o relatório de pontos por membro e tecnologia, onde é possível analisar a produtividade de cada membro de acordo com a tecnologia. Durante a obtenção dos dados somente foram realizadas tarefas utilizando duas tecnologias, devido a isto não é possível visualizar, neste momento, a diferença de produtividade de um desenvolvedor de acordo com a tecnologia. Mas pode-se observar que entres os desenvolvedores existe diferenças na produtividade.

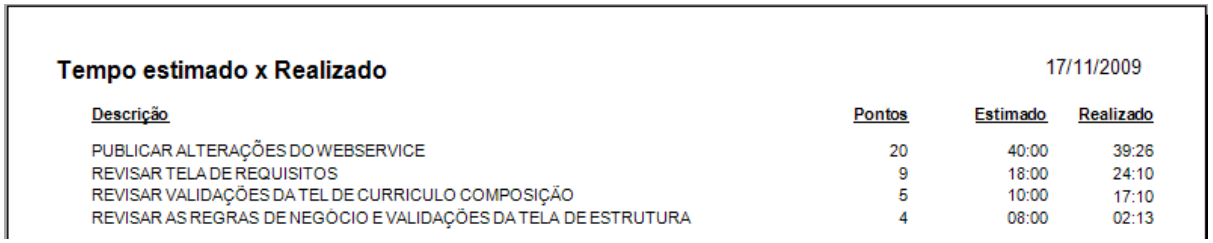


Relatório de pontos por membro e tecnologia 19/11/2009

Tecnologia		Pontos	Tempo Total	Tempo por ponto
USUÁRIO 1	**Grupo:** Desenvolvimento			
C #		9	19:23	02:90
USUÁRIO 2	**Grupo:** Desenvolvimento			
ASP.NET		20	39:26	01:58
USUÁRIO 3	**Grupo:** Testador			
C #		14	06:43	00:28
USUÁRIO 4	**Grupo:** Desenvolvimento			
C #		9	17:27	01:56

Figura 32 - Relatório de pontos por membro e tecnologia

Na figura 32, é apresentada uma comparação entre o tempo estimado para a realização da tarefa e o esforço utilizado para a sua realização. Porém como foi citado anteriormente, algumas tarefas no momento da obtenção dos dados ainda não haviam sido concluídas, sendo assim este relatório apresentada os dados do andamento de cada tarefa.



Tempo estimado x Realizado 17/11/2009

Descrição	Pontos	Estimado	Realizado
PUBLICAR ALTERAÇÕES DO WEBSERVICE	20	40:00	39:26
REVISAR TELA DE REQUISITOS	9	18:00	24:10
REVISAR VALIDAÇÕES DA TEL DE CURRICULO COMPOSIÇÃO	5	10:00	17:10
REVISAR AS REGRAS DE NEGÓCIO E VALIDAÇÕES DA TELA DE ESTRUTURA	4	08:00	02:13

Figura 33 - Tempo Estimado X Realizado

Na figura 33, são apresentadas as atividades realizadas por cada usuário e o tempo gasto com cada uma delas. Através deste relatório é possível mapear quais atividades estão tomando tempo dos membros da equipe que não são de desenvolvimento. A partir da análise realizada dos relatórios de atividades é possível observar quais os tipos de atividades que cada usuário lançou. Também é possível observar um problema encontrado durante a execução do estudo de caso, que foi como fazer as pessoas lembrarem-se de lançar as atividades.

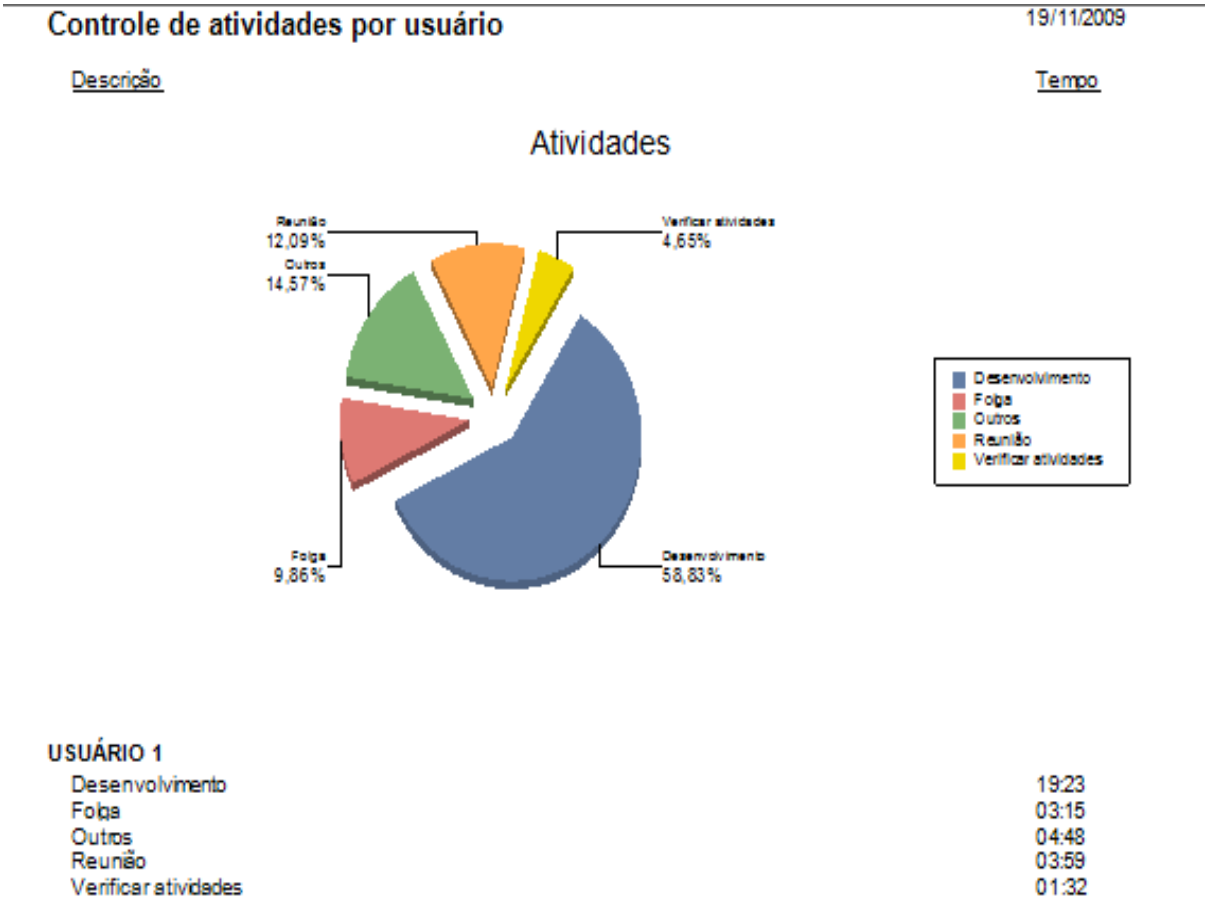


Figura 34 - Atividades por usuário

Na figura 34, é apresentado o relatório de quantidade de erros por tarefa. A partir deste relatório é possível analisar qual o tempo para a realização da tarefa e qual foi o esforço necessário para a correção de erros.

Relatório de quantidade de erro por tarefa 19/11/2009

Descrição	Pontos	Tempo	Erros	Tempo
REVISAR TELA DE REQUISITOS	9	24:10	3	07:00

Figura 35 - Relatório de quantidade de erros por tarefa

Na figura 35, é apresentado o relatório de custo por erro, este relatório tem como objetivo mostrar para o gerente qual é o custo para correção de erros encontrados pela equipe de desenvolvimento. Este relatório considera somente o esforço necessário para correção dos erros. Sendo que, para está informação é considerado o valor do custo do desenvolvedor para a empresa.

Relatório de custo por erro 19/11/2009

Descrição	Pontos	Qtd. Erro	Tempo	Custo de correção
REVISAR TELA DE REQUISITOS	9	3	07:00	R$ 112,00

Figura 36 - Relatório de custo por erro

Para a validação dos relatórios, foram apresentados os resultados para o gerente do projeto a fim de obter o parecer sobre os pontos positivos e negativos da ferramenta e seus controles. Portanto, a partir de uma conversa informal com o gerente, foi analisado que o controle da produtividade da equipe e o monitoramento dos registros de atividades são importantes, pois assim, com estas informações fica mais fácil a melhorar os processos e o aperfeiçoamento da equipe. Além disso, com o uso das informações é possível apresentar para equipe qual a importância do aprimoramento do desenvolvimento para evitar retrabalhos e custos elevados de desenvolvimento. Também, foi possível identificar melhor qual o custo de correção de erros e com esta informação é possível identificar melhor quais são os gargalos do projeto. Sendo que, como visto na figura 43, uma tarefa que levou aproximadamente 24 horas para o seu desenvolvimento até a sua conclusão, teve 3 erros, sendo que, o tempo para correção destes erros levou 7 horas, com um custo de R$ 112,00. No entanto, este custo para correção considera somente o custo do desenvolvedor não considerando o tempo que o teste levou para refazer os testes. Este seria um ponto que o trabalho deveria ser melhorado para futuramente ter o tempo e o custo total de correção e teste. Outro ponto apontado pelo gerente como sendo uma melhoria para a ferramenta, é o controle de tarefas da equipe de *helpdesk* (equipe de suporte ao sistema), que recebe solicitações diversas de seus usuários e que não tem como estimar a quantidade de pontos para a sua realização, sendo que estas atividades são geralmente criações de relatórios, manutenção ou melhorias em sistemas que não requerem uma análise para o seu desenvolvimento.

CONCLUSÃO

A partir dos estudos realizados, pode se observar que a medição e o controle da produtividade podem trazer um grande ganho para as empresas, como a melhora do tempo de desenvolvimento, a qualidade dos produtos desenvolvidos e com isso a empresa obterá a satisfação dos clientes. Como visto no capítulo 1, existem melhorias que ao longo dos anos foram tornando o desenvolvimento de *software* mais produtivo, mas ainda existem fatores que precisam ser aprimorados a cada dia para obter um equilíbrio entre tecnologia e sociologia. Para encontrar este equilíbrio na equipe, foi proposta na ferramenta uma funcionalidade para melhorar a comunicação entre as equipe, fazendo com a tarefa transite entre uma equipe e outro. Além de todo o trabalho ser registrado, permitindo o acompanhamento da tarefa.

Já com a utilização de métricas de *software*, conforme visto no capítulo 2 é possível avaliar a produtividade da equipe ou das pessoas envolvidas no projeto. As métricas apresentadas neste capítulo são algumas que a partir da evolução das linguagens de programação e com a melhoria contínua dos processos e metodologias de desenvolvimento e análise de *software*, podem ser utilizadas na medição e no controle da produtividade. As métricas foram utilizadas para encontrar uma forma de calcular a produtividade da equipe. Sendo que, para a realização deste cálculo, é utilizadas as informações de tecnologia, pontos e usuário, pois cada usuário tem a sua capacidade produtiva. Além disso, através da análise das métricas é possível identificar processos a serem melhorados ou até o aperfeiçoamento da equipe.

No capítulo 3, foram descritas as estimativas de *software*, a partir destas estima-se de forma quantitativa o tamanho e o tempo de execução de um dado projeto. Com o auxílio das estimativas, pode-se aplicar métricas, que são importantes na formação de indicadores, esse por sua vez, indicam a produtividade no desenvolvimento de sistemas.

No capítulo 4, foram apresentadas as duas formas de controle de *software* descritas por Humphrey, onde se teve como objetivo para o trabalho identificar a forma como era definida as atividades para a equipe e individual, e como é realizada a coleta dos artefatos que serão utilizados no projeto. Tendo como intuito a identificação das métricas, ou seja, de onde pode-se retirar informações, que serão utilizadas como base para a definição dos indicadores de produtividade.

A partir da análise bibliográfica sobre produtividade, propôs-se a ferramenta para controle e medição da produtividade, que tem como objetivo principal, auxiliar o gerente a controlar a produtividade de sua equipe. Além disto, a ferramenta permite o controle das tarefas que estão sendo desenvolvidas pela equipe. Com isso é possível manter um histórico de produtividade, em relação a projetos, pessoas, equipe e defeitos de projetos anteriores. Possibilitando ao gerente aperfeiçoar continuamente seus processos, métodos e o ambiente de trabalho, proporcionando às pessoas motivação e a satisfação das suas necessidades. Com isso, concluiu-se que este trabalho alcançou o objetivo proposto, permitindo um maior gerenciamento sobre a produtividade dos membros da equipe, retrabalhos, atividades não relacionadas ao projeto, oferecendo ao gerente informações sobre a capacidade produtiva da equipe. Além disso, segundo a avaliação do gerente do projeto, através da ferramenta e dos resultados apresentados é possível apresentar a equipe quais os resultados que estão sendo obtidos e procurar através de capacitações aperfeiçoar o conhecimento da equipe, a fim de aumentar a capacidade produtividade da equipe e diminuir o tempo e o custo para o desenvolvimento de *softwares*.

Como trabalho futuro pretende-se: melhor os relatórios, adicionando a ferramenta um monitoramento através de *dashboards* ou como são mais conhecidos os painéis de indicadores, que irão auxiliar o monitoramento das tarefas, projetos e a produtividade. Outro trabalho interessante seria a construção de um módulo para alocação automática das tarefas, que seria feitas através da produtividade e disponibilidade. Também será construído o gráfico de Burndown que é muito utilizado como motivador da equipe de desenvolvimento em projeto com metodologia scrun. Este gráfico será utilizado na ferramenta por ser melhor visualizado o andamento dos projetos, pela forma de visualização ser fácil e de rápida análise.

.

REFERÊNCIAS BIBLIOGRÁFICAS

AceProject, disponível em: <http://www.aceproject.com/>. Acessado em 30 de Agosto de 2009.

ANDRADE, Edméia Leonor Pereira de. **Pontos de caso de uso e pontos de função na gestão de estimativa de tamanho de projetos de software orientados a ojetos.** 2004, 143 p. Dissertação do Programa de Pós-Graduação *Stricto Sensu* em Gestão do Conhecimento e Tecnologia da Informação. Universidade Católica de Brasilia.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Campus, 2002. 286p.

ClickTime, disponível em: <http://www.clicktime.com/>. Acessado em 29 de Agosto de 2009.

DEMARCO, Tom; LISTER, Timothy. **Peopleware: como gerenciar equipes e projetos tornando-os mais produtivos**. São Paulo, SP: McGraw-Hill, 1990. 223 p.

FENTON, Norman E.; PFLEEGER, Shari Lawrence. **Software metrics: a rigorous and practical approach.** 2nd ed. Boston: PWS Publishing, [1998]. 638 p. ISBN 0-534-95425-1.

FOWLER, Martin. UML Essencial: um breve guia para a linguagem-padrão de modelagem de objetos. 3. ed. Porto Alegre: Bookman, 2005. 160p.

HUMPHREY, Watts S. **Introductio**n **to the team software process**. [1st ed.] Reading, Massachusetts: Addison-Wesley, 1999, c2000. 463 p. (The SEI series in software engineering.) ISBN 020147719X.

IFPUG, *International Function Point Users Groups*, disponível em: <http://www.ifpug.org/>. Acessado em23 maio de 2009.

Jira, disponível em: < http://www.atlassian.com/software/jira/>. Acessado em 30 de Agosto de 2009.

JONES, Capers. **Produtividade no desenvolvimento de software**. São Paulo, SP: Makron Books, McGraw-Hill, 1991. 370 p.

JONES, Capers. **Applied Software Mesurement**. 3. ed. United States, McGraw-Hill, 2008. 662 p.

KOSCIANSKI, André. **Qualidade de software**. 2 ed. São Paulo, SP: Novatec Editora, 2007. 395 p. ISBN 9788575221129.

MEDEIROS, Ernani. Desenvolvendo software com uml 2.0: definitivo. São Paulo: Pearson, 2004. 264 p. ISBN 85-346-1529-2.

PRESSMAN, Roger S. **Engenharia de software**. 5. ed. São Paulo, SP: McGraw-Hill, 2002. 843 p. ISBN 8586804258.

REZENDE, Denis Alcides. **Engenharia de software e Sistemas de informação.** 3 ed. São Paulo, SP: Brasport, 2005. 344 p. ISBN: 8574522155.

SOMMERVILLE, Ian. **Engenharia de software**. 6. ed. São Paulo, SP: Addison-Wesley, 2003. 592 p. ISBN 8588639076.

VAZQUEZ, Carlos Eduardo. **Análise de pontos de função**. 1 ed. São Paulo, SP: Érica, 2003. 230 p. ISBN 8571948992.

YOURDON, Edward**,. Análise estruturada moderna**. 1. ed. Rio de Janeiro, RJ: Campus, c1990. 836 p. (Yourdon Press) ISBN 8570016158.