Agentes e Métricas no Processo de Desenvolvimento de Software em Equipes Distribuídas

RODRIGO DE BARROS PAES[1]
HYGGO OLIVEIRA DE ALMEIDA[2]

[1]Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro
rbp@les.inf.puc-rio.br

[2]Departamento de Engenharia Elétrica, Universidade Federal de Campina Grande
hyggo@dee.ufcg.edu.br

Resumo. O uso de métricas é apontado como adequado para melhorar as estimativas e a qualidade dos softwares. Porém, o aumento da complexidade dos projetos de software, aliado a cenários onde a equipe de desenvolvimento está geograficamente distribuída, torna a utilização de métricas uma tarefa cada vez mais complicada. Este trabalho apresenta como agentes de software podem auxiliar na utilização de métricas no desenvolvimento colaborativo de aplicações. O uso de agentes tem como foco prover flexibilidade na alteração das métricas, levando em conta a natureza distribuída da equipe de desenvolvimento e a integração com ambientes de desenvolvimento já existentes.

Palavras-Chave: Métricas de software, Agentes de software, Desenvolvimento de software.

Agents and Metrics in Software Development Process for Distributed Teams

Abstract. Metrics have been pointed as a suitable mechanism for improving the software quality. However, the increasing complexity of software projects, mainly in the context of distributed teams, makes the usage of metrics a non trivial task. This work presents how software agents can be used to aid the application of metrics in the collaborative software development process. The focus is to apply agents for promoting flexibility for changes in the metrics, taking into account distributed teams and the integration with existing development environments.

Keywords: Software metrics, Software agents, Software development.

(Received February 11, 2005 / Accepted April 13, 2005)

1 Introdução

A crescente demanda pela qualidade dos serviços oferecidos por uma empresa faz com que, para ser competitiva, esta empresa necessite ter um maior conhecimento sobre o seu funcionamento e, desta forma, realizar melhorias em seu processo produtivo. No desenvolvimento de software, o uso de métricas surge como ferramenta para tornar visíveis as atividades produtivas de uma empresa. Com base nestas métricas, torna-se possível avaliar e planejar mudanças nos processos produtivos de uma maneira mais segura [29].

Em se tratando de software, considerando-se um cenário onde os desenvolvedores trabalham de forma colaborativa em um mesmo projeto e se encontram geograficamente distribuídos, a utilização de um processo de medição torna-se ainda mais complicado. Neste trabalho, apresenta-se uma arquitetura multi-agentes que dá suporte ao processo de desenvolvimento de software com equipes distribuídas, principalmente através da implantação do uso de métricas.

Na Seção 2 apresentam-se discussões relacionadas a métricas de software. Entre estas discussões estão algumas motivações para a utilização de métricas, as etapas de um processo de medição e alguns exemplos e categorias de métricas. Na Seção 3 são apresentados alguns dos principais conceitos referentes ao trabalho cooperativo. Desta forma, em conjunto com as discussões da Seção 2, são identificados os requisitos de uma solução que dê suporte ao desenvolvimento colaborativo de software e auxilie na melhoria da sua qualidade. Uma solução arquitetural utilizando uma abordagem multiagentes é proposta na Seção 4. Esta arquitetura fornece, principalmente, flexibilidade na alteração das métricas, levando em conta a natureza distribuída da equipe de desenvolvimento e a integração com ambientes de desenvolvimento existentes. Na Seção 5, a arquitetura é relacionada com outras soluções existentes na literatura. E por fim, na Seção 6, são apresentadas algumas discussões sobre este trabalho ressaltando as contribuições e trabalhos futuros.

2 Métricas

O cenário atual de alta competitividade entre as organizações faz com que a necessidade da oferta de produtos e serviços diferenciados, com qualidade e preços cada vez melhores, torne-se um fator crítico para a determinação do sucesso ou fracasso de uma organização. Neste contexto, as empresas de desenvolvimento de software vêm se preocupando, cada vez mais, com a qualidade de seus softwares. Além disso, ainda que o software atinja um nível de qualidade satisfatório ao cliente, outros fatores primordiais, tais como prazos de entrega curtos e preços acessíveis, devem ser considerados.

Uma vez que a estimativa dos prazos e custos do projeto ocorre antes do início do desenvolvimento, é imprescindível que tal estimativa seja suficientemente precisa para não acarretar prejuízos para a empresa ou para o cliente. Esta precisão é fortemente dependente da experiência do desenvolvedor que, em geral, baseia-se em medidas como número de linhas de código e número de pontos de função [24].

Entretanto, ainda que se tenha precisão na estimativa inicial, existe a possibilidade de que os prazos não sejam cumpridos durante a execução do projeto. Sendo assim, é necessário prover mecanismos para monitorar o desenvolvimento de forma a verificar se as medidas e estimativas iniciais são válidas no decorrer do projeto.

Além disso, nos casos em que os prazos não são cumpridos, também deve ser possível identificar quais são as prováveis causas dos atrasos. Isto deve ser feito o mais cedo possível, facilitando as renegociações ou mudanças na estratégia da empresa e garantindo o cumprimento dos prazos estabelecidos no contrato.

Uma das formas de acompanhar o desenvolvimento do sistema é através da realização de medições. O processo de medição pode ser visto como um conjunto de definições, métodos e atividades utilizados para medir um artefato, com o propósito de avaliar, caracterizar ou entender este artefato. A contagem de todos os *statements* segundo a gramática de uma linguagem, por exemplo, corresponde a uma medição [29].

Uma medida, por sua vez, é definida como a dimensão, capacidade, quantidade ou propriedade de um artefato, expresso em alguma unidade de medida e sempre obtido segundo uma determinada medição [29]. Exemplos de medidas são: 300 linhas de código; 7 páginas de documentação do projeto; a inteligibilidade do código é muito boa no conjunto {muito boa, boa, regular, ruim e muito ruim}.

A união entre uma unidade de medida e a correspondente medição utilizada para medir ou avaliar uma determinada propriedade de um artefato corresponde à definição de métrica.

2.1 Métricas no desenvolvimento de software

Uma vez apresentadas as definições de medição, medida e métrica, são apresentadas a seguir as razões pelas quais deve-se fazer uso destes conceitos no desenvolvimento de software. De maneira geral, as três principais motivações para medir um software são: entender o processo de engenharia de software e seus produtos de forma a derivar modelos e examinar os relacionamentos entre os elementos deste modelo; auxiliar no gerenciamento de projetos de software; e auxiliar em melhorias no processo de desenvolvimento [5].

2.1.1 Entender o processo de engenharia de software e seus produtos

Entender o que uma empresa faz e como ela faz é um requisito fundamental para qualquer tentativa de planejar, gerenciar, ou melhorar as atividades da empresa. Questões típicas que podem ser levantadas por uma empresa, cujas respostas ajudariam em um melhor entendimento de como a empresa funciona são: quanto nós estamos gastando com o desenvolvimento de software? Em que parte do ciclo de desenvolvimento os recursos estão sendo alocados? Qual o esforço demandado pela tarefa de testar o software? Quais os tipos de erros e mudanças típicas em nossos projetos?

Com o objetivo de obter respostas a estas perguntas, a medição é o único mecanismo disponível para quantificar um conjunto de características sobre um ambiente específico ou um software. Uma vez que estas medições tenham sido realizadas, torna-se possível a construção de modelos que retratam o funcionamento de uma empresa, assim como, a análise do relacionamento entre os elementos que compõem este modelo.

Entretanto, vários fatores podem contribuir negativamente para o estabelecimento de um processo de medição. Mudanças contínuas e rápidas na estrutura da empresa, a especificidade de características de cada projeto e mudanças tecnológicas muito rápidas são exemplos de tais fatores.

2.1.2 Auxiliar no gerenciamento de projetos

O conhecimento adquirido, através do uso de medições, sobre o processo de engenharia de software permitirá, entre outras coisas: estimar elementos do projeto, tais como custo e prazos; monitorar e confrontar os resultados que estão sendo alcançados com os que foram planejados; validar os modelos que foram estabelecidos para que sirvam de base para futuros projetos.

Em geral, isto é possível devido a medições resultantes de projetos similares realizados anteriormente. Tais projetos alimentam um banco de dados que é utilizado como base para derivar modelos usados para estimativas. A qualidade das informações contidas nestes bancos de dados afeta diretamente a qualidade dos modelos construídos e, conseqüentemente, a qualidade das estimativas.

Monitorar a execução do processo de desenvolvimento permite que quando as medições não ocorram como esperado as causas possam ser identificadas. A identificação das causas é possível através do relacionamento entre os elementos do modelo utilizado no processo de medição. Embora o fato de se obter uma medida diferente da esperada não signifique necessariamente um problema, através desta identificação, o desenvolvedor estará apto a avaliar o seu significado.

2.1.3 Auxiliar em melhorias no processo de desenvolvimento

Um dos objetivos principais de uma organização é produzir produtos de qualidade, dentro dos prazos esperados e com um preço acessível. Uma das formas de se conseguir uma melhoria na qualidade de um produto é melhorar a forma como ele é feito. Porém, ao introduzir qualquer mudança, é necessário "medir" e verificar as reais conseqüências provocadas por esta mudança.

Existem várias abordagens voltadas para o projeto e a introdução de métricas visando o melhoramento do processo de desenvolvimento de uma empresa. O *Capability Maturity Model* (CMM) [28] é um exemplo deste tipo de abordagem. O CMM é um modelo para julgar a maturidade de um processo de desenvolvimento de empresas de software. Este modelo é usado como um auxílio na identificação das principais práticas necessárias para melhorar os seus processos de desenvolvimento. Além do CMM, outros modelos existentes podem ser citados, tais como o CMMI [17] e o MPS-Br [33].

2.2 Etapas de um processo de medição

Uma vez que as motivações para o uso de métricas já foram discutidas, o próximo passo é a implantação de um processo de medição. Porém, antes de implantar este processo é necessário analisar os custos desta implantação e verificar se esta implantação realmente trará benefícios.

Em geral, é possível identificar ao menos três papéis em um processo de medição: o detentor das informações sobre o desenvolvimento do projeto; o analista que examina as medições, deriva modelos de processos e relacionamentos entre os elementos deste processo; e, finalmente, o papel de suporte, que coleta e armazena as informações para futuras consultas.

Os custos da implantação de um processo de medição estão relacionados a estes três papéis, seja através: da realização da tarefa de elaboração de formulários de responsabilidade do papel de analista; do preenchimento destes formulários, responsabilidade dos papéis de detentor e suporte; do encontro com os analistas, responsabilidade de todos os papéis; e da identificação dos objetivos da empresa e análise dos dados que são de responsabilidade do papel de analista.

A decisão de quanto é o máximo de custo tolerado na implantação de um processo de medição no custo do desenvolvimento de um software é dependente de cada empresa. Entretanto, em um trabalho [5] publicado pelo Laboratório de Engenharia de Software da NASA (*National Aeronautics and Space Administration*), sugere-se que os custos com medições não devem adicionar mais do que: 2% com custos de desenvolvimento e manutenção (detentor); 7% com custos com o suporte técnico (suporte); e 15% com a análise das métricas (analista).

Uma vez que se opta por utilizar métricas no processo de desenvolvimento de uma empresa, é necessário um maior entendimento de quais são as etapas que compõem este processo. No trabalho de Andrews [25], são definidas três macro-fases necessárias à maioria dos processos de medição. Estas três fases são descritas a seguir:

- *Planejamento* - nesta fase, procura-se identificar os objetivos da medição, quais os assuntos e problemas envolvidos para realizá-la, onde ela será aplicada (processos, produtos etc) e quais são as métricas que auxiliam a responder os objetivos;
- *Implementação* - nesta fase, de acordo com as métricas definidas na etapa de Planejamento, os dados são coletados, armazenados e analisados. A saída desta fase é usada como entrada para a etapa de Melhoria;
- *Melhoria* - o processo é avaliado, bem como o progresso no desenvolvimento do software. Decisões devem ser tomadas visando melhorar o processo de desenvolvimento.

Estas fases devem ser executadas iterativamente. As iterações podem ocorrer dentro do mesmo projeto ou em projetos diferentes através do conhecimento obtido em projetos anteriores. Desta forma é possível:

- coletar medidas baseadas no histórico dos projetos já finalizados para ajudar a entender, por similaridade, o processo de software atual;

- usar os históricos para calibrar os modelos de estimativa de software;
- desenvolver estimativas coerentes e planejar o tamanho, esforço e agenda dos futuros projetos;
- confrontar os dados que estão sendo coletados com os que foram planejados para saber quão completo está o projeto, quanto ainda precisa ser feito, quando o projeto estará completo etc;
- replanejar o projeto baseado nestes indícios de estado e progresso e até mesmo renegociar os requisitos.

Cada uma destas fases pode ser composta de outras atividades que não foram descritas neste trabalho. Porém, as métricas obtidas na execução de parte destas atividades tornam os processos de software e os produtos mais visíveis de forma que uma empresa possa melhorar a tomada de decisão e gerenciar os seus objetivos.

De forma geral, em uma visão micro destas fases, pode-se identificar como atividades-chave: a coleta de métricas; o armazenamento e visualização destas métricas; e a análise feita a partir da visualização destas métricas.

Em relação à atividade de coleta, as métricas podem ser capturadas basicamente através de formulários impressos, ferramentas automáticas e entrevistas pessoais, sendo este último geralmente utilizado para capturar alguma informação subjetiva. Além disso, para que esta atividade seja bem sucedida, é preciso que sejam coletados os dados "corretos" e que haja dados suficientes para uma futura análise. De acordo com Zelkowiz [36], os seguintes aspectos, dentre outros, precisam ser considerados na atividade de coleta de dados:

- *Replicação* - quando a coleta de dados é realizada para validação de alguma tecnologia, é importante possibilitar a replicação dos resultados de um experimento de forma que outros pesquisadores também possam reproduzir os resultados;
- *Influência do contexto* - no desenvolvimento de software pode ser difícil identificar todos os fatores que influenciam nos resultados de uma coleta de dados. Usualmente, um conjunto de pessoas, ferramentas, métodos e artefatos estão envolvidos no desenvolvimento de software e todos eles podem influenciar na atividade de coleta;
- *Propriedades temporais* - os dados podem ser históricos ou atuais e é preciso levar em consideração os aspectos relacionados ao momento em que foram coletados. Desta forma, a análise pode considerar aspectos externos à coleta no momento de interpretar os dados.

Uma vez que os dados foram coletados, é preciso sintetizá-los de forma a facilitar a análise destas informações. Esta etapa de síntese consiste em agrupar os dados em gráficos, tabelas, valores e outras maneiras que permitam analisar mais eficientemente os resultados obtidos.

A etapa de análise é certamente a atividade mais importante de um processo de medição. É nesta atividade que as métricas são analisadas e, com base nos resultados desta análise, é possível planejar mudanças, avaliar tecnologias e todos os demais itens que foram descritos anteriormente.

2.3 Categorias e exemplos de métricas

Uma etapa importante na implantação de um processo de medição é a escolha das medidas. Esta escolha deve ser feita sempre visando alcançar os objetivos da organização na qual as métricas serão aplicadas. De maneira geral, as medidas podem ser objetivas ou subjetivas e são classificadas mediante a sua aplicabilidade. Esta classificação pode ser organizada em uma hierarquia, onde as categorias de aplicação mais genéricas ocupam posições mais altas nesta hierarquia. Na Figura 1, ilustra-se uma possível classificação de métricas. Nos nós externos da hierarquia é possível observar que a categoria de métricas de software possui como "nós-filhos" as categorias mais específicas de métricas para produto, processo e recursos (pessoas, materiais, ferramentas etc.)

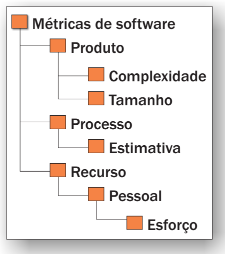

Figura 1: Exemplo de uma classificação de métricas

Não há um consenso em relação a uma taxonomia genérica de métricas. Porém, algumas taxonomias foram propostas considerando categorias de aplicação mais específicas, como por exemplo, taxonomia de métricas de desempenho [35] e de orientação a objetos [9].

Para que as métricas possam ser reutilizadas e os resultados de sua aplicação em diferentes projetos possam ser comparados é necessário definir a forma como estas métricas são coletadas, representadas e analisadas. A definição das métricas de maneira não-ambígua e precisa é um dos principais problemas da utilização de um processo de medição [20].

Alguns paradigmas existentes, tais como o GQM (*Goal-Question-Metrics*) [4], são importantes para que as métricas sejam úteis, simples e diretas. Entretanto, nestes paradigmas as métricas não são definidas no nível de detalhes necessário para garantir confiabilidade. Em particular, não é explicitado se as métricas podem ou não ser repetidas, ou seja, se a medição de um atributo for repetida por uma pessoa diferente, o mesmo resultado deve ser obtido.

Um exemplo desta ambigüidade pode ser encontrado nas definições da métrica de linhas de código de um software. Jones, em [19], apresentou as seguintes variações: contar somente as linhas executáveis; contar as linhas executáveis e as definições de dados; contar as linhas executáveis, as definições de dados e os comentários; contar as linhas que aparecem no monitor; contar somente as linhas terminadas por delimitadores lógicos.

Várias outras definições para esta métrica relativamente simples podem ser especificadas e, portanto, há uma necessidade de se estabelecer um padrão de especificação de métricas que permita expressar uma métrica com detalhes suficientes para torná-la não ambígua e que ao mesmo tempo seja de fácil especificação. Neste sentido, no trabalho de Kitchenham [20] é proposto um modelo baseado em uma abordagem Entidade- Relacionamento que permite a modelagem e o armazenamento de métricas de software. Além destas características de especificação e armazenamento de métricas apresentadas nesta seção, o restante desta seção apresenta alguns exemplos de métricas.

No trabalho publicado em [13], sugere-se que as métricas sejam categorizadas por tamanho, esforço e planejamento, qualidade, desempenho, confiabilidade e complexidade. Para cada uma destas categorias é proposto um conjunto de métricas que são agrupadas em classes de atributos relacionados ao software. Um exemplo destas classes de atributos é descrito na tabela ilustrada na Figura 2. Nesta tabela, estão relacionadas as métricas de tamanho de software agrupadas em nove classes de atributos de um software. Além destas categorias descritas acima, várias outras categorias de métricas podem ser encontradas na literatura [5, 8] . Cada uma destas categorias se aplica a contextos bem específicos e a definição das métricas também é bastante variada.

Através das classes de atributos representadas na Figura 2, é possível definir vários tipos de métricas de tamanho. Para isto, selecionam-se as classes e os itens de descrição de cada classe que devem ser considerados. Este mesmo raciocínio é aplicado às outras categorias de métricas, tais como qualidade e complexidade. Um exemplo de métrica utilizando essa classificação é a métrica de *Linhas de Código Físicas*, também definida em [13]. Nesta métrica, deve-se contar todos os comandos executados, qualquer tipo de declaração e diretivas de compilação. Esta métrica também classifica como as linhas foram produzidas e, para isto, calcula a partir do total de linhas físicas: quantas foram programadas; quantas foram produzidas por geradores automáticos; quantas foram convertidas por tradutores automáticos; quantas foram copiadas ou reutilizadas sem nenhuma modificação; e quantas foram modificadas de uma versão anterior de um mesmo programa.

Classe de atributo	Descreve
tipo de comando	comandos executáveis e não executáveis, e se o comando é uma declaração, um comentário, uma diretiva do compilador ou uma linha em branco;
como foi produzido	código programado por um engenheiro de software, criado por uma ferramenta de geração de código, convertido de um outro programa ou linguagem por um tradutor automático, copiado ou reutilizado sem modificações de um outro programa, modificado de uma versão anterior de um mesmo programa ou removido de uma versão anterior;
originalidade	novo trabalho ou trabalho já realizado;
uso	o código é parte do produto principal ou serve de suporte ao produto principal;
entrega	o código precisa ser entregue a alguém ou é usado internamente;
funcionalidade	operante ou inoperante;
replicações	o código precisa ser replicado durante a compilação ou durante a instalação;
status de desenvolvimento	onde está o código em relação ao processo de desenvolvimento: planejado, projetado, codificado, testes de unidade completados etc;
linguagem	a linguagem de programação utilizada.

Figura 2: Categorias de métricas de tamanho de software [13]

3 Trabalho cooperativo

A Engenharia de Software vem utilizando a Internet como meio para conduzir as atividades de um processo de desenvolvimento de software de maneira distribuída [22]. O desenvolvimento das atividades de forma distribuída possui uma série de vantagens: os membros de um projeto podem estar geograficamente distribuídos, eliminando custos com transporte e acomodação; as pessoas com habilidades adequadas a um projeto podem trabalhar em conjunto independentemente de suas localizações; e os membros do projeto podem trabalhar em suas próprias casas, diminuindo assim os gastos da empresa.

Do ponto de vista do desenvolvedor, esta forma de trabalhar significa que ele pode estar trabalhando em projetos diferentes, com equipes diferentes, usando processos diferentes e tudo isso ao mesmo tempo. Desta forma, é preciso que haja uma infra-estrutura computacional que forneça suporte ao gerenciamento deste cenário.

No restante desta seção é apresentada uma visão geral sobre as abordagens utilizadas para prover suporte ao desenvolvimento de software de maneira distribuída. Com isto, são levantados os requisitos para o desenvolvimento da arquitetura multi-agentes proposta.

3.1 Terminologia

Quando os membros de uma equipe trabalham no desenvolvimento de um software, estão compartilhando um objetivo em comum: concluir o desenvolvimento do software. Isto significa que estes membros estão realizando um trabalho cooperativo. Porém, além de trabalharem de forma cooperativa, é necessário que as atividades sejam coordenadas para a correta execução do projeto. Coordenação é definida como a gerência das dependências entre as atividades [23].

Várias áreas de pesquisa lidam com assuntos relacionados ao trabalho cooperativo auxiliado por mecanismos computacionais. No trabalho de Bischofberger [6] são apresentadas três abordagens que, de alguma forma, lidam com o trabalho cooperativo. Uma breve síntese de cada uma destas abordagens é apresentada a seguir:

- *Engenharia de Software Cooperativa* - compreende todos os métodos, normas e ferramentas de engenharia de software que auxiliam o trabalho em equipe de forma flexível e eficiente. Portanto, a engenharia de software cooperativa é um subconjunto da engenharia de software. Esta engenharia fornece mecanismos de cooperação baseada em políticas e cooperação informal. A cooperação baseada em políticas é regimentada por um acordo formal, o modelo de coordenação é implementado através da troca de documentos bem estruturados e é auxiliada por ferramentas que gerenciam a evolução e gerenciamento destes documentos. Já na cooperação informal, não existe um regimento. A informação trocada não precisa ser estruturada e é auxiliada por tudo que lida com comunicação, como por exemplo serviços de email;

- *Trabalho Cooperativo Apoiado por Computador*[1] *(TCAC)* - o objetivo desta abordagem é auxiliar grupos na colaboração e coordenação de suas atividades [2, 3]. Uma taxonomia para os tipos de cenários tratados nesta abordagem é descrita na tabela ilustrada na Figura 3;

	Ao mesmo tempo	**Instante diferente**
Mesmo lugar	interação face-a-face	interação assíncrona
Lugar diferente	interação síncrona e distribuída	interação assíncrona e distribuída

Figura 3: Taxonomia de cenários tratados pela TCAC [10]

- *Engenharia de Software Centrada em Processo.* Nesta abordagem tenta-se estabelecer uma base teórica para o entendimento, descrição e execução de processos de desenvolvimento de software [26]. Em outras palavras a idéia básica é ter uma representação explícita do processo que especifica como as atividades do desenvolvimento de software devem ser realizadas, bem como os papéis e tarefas dos desenvolvedores e como usar e controlar as ferramentas de software. O modelo de processo resultante, entre outras coisas, define em qual seqüência quais documentos podem ser acessados e por qual ferramenta.

[1] Esta abordagem é mais conhecida por *Computer Supported Cooperative Work*

3.2 Identificação de requisitos

De acordo com a Engenharia de Software Cooperativa, a cooperação pode ocorrer de maneira formal ou informal. Desta forma, uma ferramenta que dê suporte à esta engenharia deve fornecer mecanismos para contemplar estas duas formas de cooperação. Ou seja, deve haver mecanismos que permitam especificar um processo de desenvolvimento com todas as suas atividades, assim como a coordenação destas atividades e a forma de comunicação entre os participantes, seja esta comunicação estruturada ou não.

Weinreich [34] apresenta um ambiente para o desenvolvimento de software de forma cooperativa. Este ambiente tenta combinar duas visões: a visão do produto e a visão do processo. Na primeira, o software é visto como um conjunto de artefatos tais como documentos de especificação e projeto, manuais, código fonte etc. Na segunda visão, foca-se em como os membros da equipe de desenvolvimento trabalham juntos, como os artefatos podem ser construídos cooperativamente e como eles trocam informações sobre as modificações ocorridas. Um dos requisitos de um ambiente que dê suporte ao desenvolvimento cooperativo é possuir uma visão individual e coletiva do processo de desenvolvimento.

Desta forma, o processo juntamente com as atividades deve poder ser visto por todos os desenvolvedores na visão coletiva. Porém, somente alguns destes desenvolvedores terão acesso completo a cada uma das atividades.

4 Arquitetura multi-agentes

4.1 Requisitos para medição em ambiente cooperativo

Com base nos requisitos de trabalho cooperativo apresentados na Seção 3.2 e nas discussões sobre métricas apresentadas na Seção 2, foi identificado um conjunto de requisitos necessários a um software que dê suporte à implantação de um processo de medição em um ambiente de trabalho cooperativo. Estes requisitos estão descritos a seguir:

- deve ser possível especificar o processo de desenvolvimento juntamente com as suas atividades, seja esta especificação formal ou informal;

- deve-se monitorar qual atividade do processo está sendo executada em um determinado momento;

- os membros das equipes de desenvolvimento devem ser tratados de forma personalizada pelo ambiente;

- os desenvolvedores podem estar geograficamente distribuídos;

- a forma de coletar as métricas deve ser flexível para permitir a adição de novos algoritmos de coleta e a modificação dos existentes;

- a arquitetura deve possibilitar a adição de mecanismos que utilizem os resultados das métricas, como mecanismos de análise de métricas;

- deve ser possível configurar alertas de acordo com o resultado das medições, notificando resultados bons ou ruins;

- devem existir ferramentas de comunicação que possibilitem a interação síncrona e assíncrona entre os desenvolvedores do projeto, como ilustrado na Figura 3;

- um dos problemas encontrados na medição de software é que as medições raramente são inteiramente suportadas pelas ferramentas de desenvolvimento de forma automática [31]. Por outro lado, foi observado que a automação da coleta e análise dos dados medidos traz significantes benefícios quando são comparados a uma alternativa manual [21, 27]. Sendo assim, a coleta dos dados deve ser automatizada sempre que possível. Porém, também deve ser permitida a inserção manual de medidas.

4.2 Cenário de aplicação

Com base nos requisitos apresentados anteriormente, a seguir é apresentado um cenário de uso deste ambiente de software, onde pretende-se ilustrar algumas destas funcionalidades. Neste cenário, os desenvolvedores utilizam as ferramentas de comunicação que estão inseridas no ambiente, tais como bate-papo e videoconferência, para discutir questões referentes ao projeto, como por exemplo, a modelagem de uma determinada parte do sistema. Enquanto isso, um elemento coletor de métricas, que também está embutido no ambiente, monitora o desenvolvedor e extrai métricas de produtividade e esforço.

Porém, para extrair as métricas de produtividade, este elemento precisa interagir com outro elemento que tem como funcionalidade a extração de informações do código fonte de um programa. O resultado destas métricas é, por sua vez, analisado por um outro elemento, denominado Analisador. O Analisador pode utilizar o resultado desta análise para alertar automaticamente um determinado desenvolvedor sobre a sua produtividade. Este elemento também pode interagir com outros elementos analisadores localizados nos ambientes de outros desenvolvedores. Esta interação pode ser útil para comparar os resultados das análises e identificar como está a produtividade de toda a equipe.

Além disso, outros elementos podem implementar algoritmos de aprendizagem, permitindo que os modelos de estimativas possam ser automaticamente ajustados. Algoritmos de aprendizagem mais sofisticados podem ser usados para auxiliar o desenvolvedor evitando, por exemplo, que cometa erros reincidentes. O conhecimento adquirido com um desenvolvedor pode ser compartilhado com elementos localizados em outros ambientes.

O processo de desenvolvimento também deve poder ser especificado para que assim cada ambiente, obtendo elementos capazes de interpretar este processo, possa identificar o andamento da execução do projeto. Para isso, é necessário saber o estágio de desenvolvimento para cada desenvolvedor, caracterizando assim, mais uma vez, a interação entre elementos pertencentes a ambientes distintos.

O cenário descrito anteriormente não tem a pretensão de abordar todas as questões que podem surgir no desenvolvimento colaborativo de software. Sendo assim, a solução computacional para este problema deve ser suficientemente flexível para que novas funcionalidades possam ser adicionadas. Além disso, em se tratando de um cenário onde os vários desenvolvedores estão distribuídos e utilizam ambientes de desenvolvimento também distribuídos, é necessário que a solução também forneça uma infra-estrutura de comunicação que dê suporte à distribuição.

4.3 Agentes e métricas

A abordagem multi-agentes vem sendo considerada adequada para lidar com sistemas que são compostos de muitos elementos, provavelmente distribuídos e desenvolvidos por equipes diferentes, que precisam interagir, freqüentemente seguindo complexos protocolos de interação [18]. Parte desta adequação se deve ao fato de que a abstração de agentes já incorpora conceitos como autonomia, interação baseada em uma linguagem compartilhada por todos os agentes e representação do conhecimento.

Autonomia é a habilidade do agente para agir sem intervenção externa, dizer sim ou não em resposta a uma requisição de outro agente e iniciar uma ação por motivações próprias [15].

Um mecanismo de interação entre dois sistemas é composto por uma linguagem e por um meio de comunicação que a transmite. Uma possível solução para a definição do mecanismo de interação entre os elementos de um sistema é cada subsistema definir o mecanismo pelo qual outros subsistemas interagem com ele. Neste caso, cada subsistema define a sintaxe e a semântica de interação, as quais devem ser compreendidas pelos sistemas interessados em interagir com ele. Na medida em que cresce a quantidade de subsistemas e, conseqüentemente, a quantidade de definições de interação sintáticas e semânticas, tem-se um maior esforço para integrar os subsistemas. Por outro lado, o uso de uma linguagem compartilhada para a interação entre os elementos de um sistema funciona como um padrão de sintaxe e semântica para todos os elementos, permitindo que o crescimento no número de elementos não implique na compreensão de novas definições sintáticas e semânticas por parte dos sistemas existentes. A abordagem multi-agentes utiliza este tipo de linguagem para realizar as interações entre os agentes. Além disso, frequentemente estas linguagens são independentes da linguagem de programação que os agentes foram desenvolvidos, permitindo assim, que agentes heterogêneos possam se comunicar. Exemplos de tais linguagens são FIPA-ACL [12] e KQML [11].

Quando dois elementos interagem, eles trocam informações ou conhecimento sobre um determinado conceito. É necessário que haja um consenso em relação ao significado e representação deste conceito para evitar ambigüidades. A utilização de ontologias para a especificação dos conceitos compartilhados pelos elementos provê maior facilidade de integração destes elementos. Desta forma, novos elementos podem ser integrados desde que utilizem o padrão de representação e conceitualização da ontologia. O uso de ontologias também é incorporado na abordagem multi-agentes, principalmente através de uso de linguagens como DAML +OIL [1] e KIF [14].

Desta forma, o uso de linguagens de comunicação compartilhadas e independentes da linguagem na qual o agente foi programado permite uma maior heterogeneidade e escalabilidade do sistema. A utilização de ontologias faz com que as ambiguidades sejam eliminadas em sistemas que possuam um grande número de conceitos envolvidos ou domínios de conhecimento e a autonomia faz com que a representação dos comportamentos dos desenvolvedores sejam melhor capturados. Estas características são adequadas ao presente sistema uma vez que vários domínios de conhecimento podem existir e interagir entre si, tais como métricas, testes e depuração.

Utilizando-se uma abordagem multi-agentes, cada um dos elementos que compõem o ambiente pode ser visto como um agente. Sendo assim, estes elementos intrinsicamente possuem baixo acoplamento e distribuição, garantidos pela abstração de agente no contexto de sistemas multi-agentes [18]

O desenvolvedor interage com o ambiente através de ferramentas que implementam funcionalidades, tais como editor de código, depurador, coletor de métricas etc. Entretanto, as funcionalidades destas ferramentas podem ser providas por um ou mais agentes, que interagem para compor estas funcionalidades. Além disso, os agentes situados em um ambiente de desenvolvimento podem interagir com agentes localizados em outros ambientes para compor as funcionalidades das ferramentas. Desta forma, na Figura 4 é apresentada uma arquitetura multi-agentes que contempla os requisitos de flexibilidade vistos até aqui. Nesta figura, estão ilustradas algumas ferramentas que compõem um ambiente. O desenvolvedor utiliza estas ferramentas para interagir com os outros membros da equipe de desenvolvimento e também para realizar as tarefas necessárias a execução do projeto. Para compor a funcionalidade de uma ferramenta, os agentes interagem com agentes situados no mesmo ambiente ou com agentes localizados nos ambientes dos outros desenvolvedores.

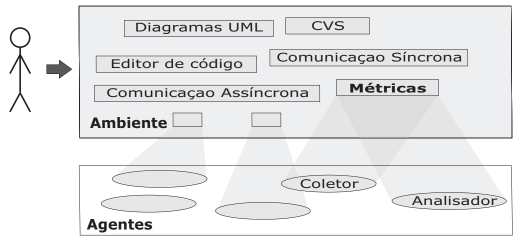

Figura 4: Arquitetura do ambiente

4.4 Integração ao ambiente Eclipse

O ambiente Eclipse [16] possui uma arquitetura flexível que possibilita a sua extensão através do uso de *plug-ins*, os quais permitem adicionar funcionalidades ao ambiente. Através destes *plug-ins*, várias funcionalidades são providas, tais como, edição de código fonte, depuração e várias outras ferramentas necessárias ao desenvolvimento de software.

O ambiente é baseado em perspectivas de um determinado projeto. Por exemplo, no desenvolvimento de um sistema voltado para a Internet, vários desenvolvedores de domínios de conhecimento diferentes atuam no mesmo projeto. O administrador do banco de dados está preocupado em como os dados serão armazenados, enquanto o desenvolvedor JAVA se preocupa em tornar o código flexível e o *designer* WEB se preocupa com a comunicação visual do *site*. O Eclipse lida com estes diferentes pontos de vista entre os desenvolvedores através do conceito de perspectivas. Uma perspectiva mostra uma visão especializada do projeto, permitindo ao administrador de banco de dados abrir a sua perspectiva para visualizar as tabelas dos bancos de dados e um conjunto de ferramentas que auxiliem o seu trabalho. O uso de perspectivas permite contemplar o requisito de se ter visões personalizadas de um projeto, como descrito na Seção 3.2.

Desta forma, tendo o Eclipse como o ambiente de desenvolvimento onde os agentes irão "habitar", é necessário estender este ambiente para que isto ocorra. Esta extensão é possível considerando-se cada agente como um *plug-in*. O fato de serem *plug-ins*, dá a estes agentes a possibilidade de estender o ambiente Eclipse fornecendo novas perspectivas, novas visões e de muitas outras formas como pode ser visto em [30].

De forma mais concreta, uma instância do trabalho poderia ser composta de três tipos de agentes com as seguintes responsabilidades:

- *Papel de auxiliar a coordenação ao desenvolvimento* - estes agentes controlam a execução de um processo de desenvolvimento, acompanhando a execução de atividades. Mais especificamente, estes agentes controlam os agentes coletores, indicam a utilização de agentes auditores e gerenciam a evolução dos testes;

- *Papel de auditor* - Estes agentes auxiliam na auditoria de testes automatizados e na comparação de métricas coletadas pelo agente coletor com métricas associadas a metas de qualidade no desenvolvimento a construção do software. A auditoria dos testes automatizados é realizada dinamicamente, por exemplo, através de uma análise sintática, e através da automação de parte das atividades de testes, por exemplo, executando testes de unidade automaticamente;

- *Papel de coletor de informações* - agentes desempenhando este papel, coletam informações sobre o histórico de testes efetuados (ex: erros/unidade) e métricas de acompanhamento do projeto (ex: horas trabalhadas/linhas de código produzidas). Estas informações são então, requisitadas pelos outros agentes.

5 Arquiteturas relacionadas

Algumas arquiteturas já foram propostas para auxiliar no desenvolvimento de software de forma colaborativa. Algumas delas focam na construção do próprio ambiente de forma flexível, como o trabalho proposto em [7]. Neste trabalho, um modelo de componentes é proposto de forma a tornar a adição e o gerenciamento dos elementos do ambiente mais flexível. Porém, a arquitetura apresentada na Seção 4 também é baseada em um modelo de componentes fornecido pelo ambiente Eclipse, através do uso de *plug-ins*. Já no trabalho proposto por Wang [32], uma arquitetura multi-agentes é apresentada como solução ao desenvolvimento colaborativo de software. Porém, nesta abordagem, não existe a integração desta arquitetura com um ambiente de desenvolvimento.

6 Discussões

As discussões sobre métricas e atividades relacionadas apresentadas na Seção 2 servem como base para a identificação dos requisitos e para o projeto de agentes, os quais auxiliam na automatização de muitas etapas de um processo de medição. Além disso, a arquitetura multi-agentes mostrada na Seção 4 apresenta-se como uma solução flexível para a implantação de uma infraestrutura de software que dê suporte ao desenvolvimento de software de maneira colaborativa.

O uso do ambiente Eclipse para o desenvolvimento traz como vantagens a sua ampla utilização e um modelo de componentes que permite grande flexibilidade nas funcionalidades providas pelo ambiente. A integração de mecanismos de melhoria na qualidade de software a este tipo de ambiente é indispensável à utilização em larga escala de tais mecanismos.

Referências

[1] The darpa agent markup language homepage, August 2000.

[2] Acm 2002 conference on computer supported cooperative work, November 2002.

[3] Ecscw'03 - 8th european conference of computer-supported cooperative work, 2003.

[4] Basili, V. R., Caldiera, G., and Rombach, H. Goal question metric paradigm. In Marciniak, J. J., editor, *Encyclopedia of Software Engineering*, pages 528–532. John Wiley & Sons, 1994.

[5] Bassman, M. J., McGarry, F., and Pajerski, R. Software measurement guidebook - revision 1. Software Engineering Laboratory Series - National Aeronautics and Space Administration, June 1995.

[6] Bischofberger, W. R., Kofler, T., Mätzel, K.-U., and Schäffer, B. Computer supported cooperative software engineering with beyond-sniff. In *Proceedings 1995 Software Engineering Environments*, pages 135–43, Noordwijkerhout, The Netherlands, 5–7 1995. IEEE.

[7] Biuk-Aghai, R. P. Customizable software engineering environments for flexible distributed software teams. In *Proceedings of Asia Pacific Software Engineering Conference*, pages 228–235, Taipei, Taiwan, R.O.C., Dec. 2–4, 1998.

[8] Bluemke. Object oriented metrics a survey. In *Proceedings of 26th ASU conference Object Oriented Modelling and Simulation*, pages 43–53, Malta, 2000.

[9] Brito e Abreu, F. and Carapuça, R. Candidate metrics for object-oriented software within a taxonomy framework. *The Journal of Systems and Software*, 26(1):87–96, 1994.

[10] Ellis, C. A., Gibbs, S. J., and Rein, G. Groupware: Some Iissues and Experiences. *Communications of ACM*, 34(1):39–58, 1991.

[11] Finin, T., Fritzson, R., McKay, D., and McEntire, R. KQML as an Agent Communication Language. In Adam, N., Bhargava, B., and Yesha, Y., editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.

[12] for Intelligent Physical Agents, F. F. Fipa acl message structure specification, December 2002.

[13] Ford, G. Software engineering measures. In *Lecture Notes on Engineering Measurement for Software Engineers*, number CMU/SEI-93-EM-9. SEI - Carnegie Mellon University, 1993.

[14] Genesereth, M. R. Knowledge interchange format. Draft proposed American National Standard - NCITS.T2/98-004, 1998.

[15] Huhns, M. N. and Singh, M. P. Agents and multiagent systems: Themes, approaches, and challenges. In Huhns, M. N. and Singh, M. P., editors, *Readings in Agents*, chapter 1, pages 1–23. Morgan Kaufmann, October 1997.

[16] Inc., O. T. I. Eclipse plataform technical overview, February 2003.

[17] Institute, S. E. Capability maturity model integration. http://www.sei.cmu.edu/cmmi/. Acessado em 25/04/2005, 2005.

[18] Jennings, N. R. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.

[19] Jones, C. *Programming Productivity*. McGraw-Hill, Inc., 1986.

[20] Kitchenham, B. A., Hughes, R. T., and Linkman, S. G. Modeling Software Measurement Data. *IEEE Trans. Softw. Eng.*, 27(9):788–804, 2001.

[21] Komi-Sirviö, S., Parviainen, P., and Ronkainen, J. Measurement automation: Methodological background and practical solutions-a multiple case study. In *Seventh International Software Metrics Symposium*, London, England, April 2001. IEEE Press.

[22] Lanubile, F. and Mallardo, T. Tool support for distributed inspection. In *26th Annual International Computer Software and Applications Conference*, Oxford, England, August 2002. IEEE press.

[23] Malone, T. W. and Crowston, K. The Interdisciplinary Study of Coordination. *ACM Comput. Surv.*, 26(1):87–119, 1994.

[24] Matson, J. E., Barrett, B. E., and Mellichamp, J. M. Software development cost estimation using function points. *IEEE Trans. Softw. Eng.*, 20(4):275–287, 1994.

[25] McAndrews, D. R. Establishing a software measurement process. Technical report, Software Engineering Institute Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, July 1993.

[26] Osterweil, L. J. Software processes are software too, revisited: an invited talk on the most influential paper of icse 9. In *Proceedings of the 19th international conference on Software engineering*, pages 540–548. ACM Press, 1997.

[27] Parviainen, P., Järvinen, J., and Sandelin, T. Practical Experiences of Tool Support in a GQM-based Measurement Programme. *Software Quality Journal*, 6(4):283–294, 1997.

[28] Paulk, M. C., Curtis, B., Chrissis, M. B., and Weber, C. V. Capability Maturity Model, Version 1.1. *IEEE Software*, 10(4):18–27, July 1993.

[29] Pressman, R. S. *Software Engineering: A Practitioner's Approach - 5a edição*. McGraw-Hill Higher Education, 2002.

[30] Shavor, S., D'Anjou, J., Fairbrother, S., Kehn, D., Kellerman, J., and McCarthy, P. *The Java Developer's Guide to Eclipse*. Addison-Wesley Pub Co, May 2003.

[31] Vierimaa, M., Tihinen, M., and Kurvinen, T. Comprehensive approach to software measurement. In *4th European Conference on Software Measurement and ICT Control*, pages 237–246, Heidelberg - Germany, May 2001.

[32] Wang, A. I., Conradi, R., and Liu, C. A multiagent architecture for cooperative software engineering. In *The Eleventh International Conference on Software Engineering and Knowledge Engineering (SEKE'99)*, pages 1–22, Kaiserslautern, Germany, June 1999.

[33] Weber, K., Rocha, A. R., Ângela Alves, Ayala, A. M., Gonçalves, A., Paret, B., Salviano, C., Machado, C. F., Scalet, D., Pelit, D., Araújo, E., Girão, M., Oliveira, K., Oliveira, L., Amaral, M., Endriss, R., and Maciel, T. Modelo de referência para melhoria de processo de software: uma abordagem brasileira. In Solar, M., Fernández-Baca, D., and Cuadros-Vargas, E., editors, *30ma Conferencia Latinoamericana de Informática (CLEI2004)*, pages 461–476. Sociedad Peruana de Computación, Sept. 2004.

[34] Weinreich, R. and Altmann, J. An object-oriented infrastructure for a cooperative software development environment. In *5th International Symposium on Applied Corporate Computing (ISACC 97)*, Monterrey, Mexico, November 1997.

[35] Worlton, J. Toward a Taxonomy of Performance Metrics. *Parallel Computing*, 17(10–11):1073–1092, Dezembro 1991.

[36] Zelkowitz, M. V. and Wallace, D. Experimental Models For Validating Technology. *Computer*, 31(5):23–31, May 1998.