Uma Arquitetura para o Gerenciamento de Artefatos de Testes Desenvolvidos por Equipes Distribuídas

Tatiane Macedo Prudencio Lopes, Clovis Torres Fernandes

Instituto Tecnológico de Aeronáutica (ITA)
Pça. Marechal Eduardo Gomes, 50, CEP. 12228-900 Vila das Acácias

São José dos Campos – SP - Brasil

tatiane@ita.br, clovistf@uol.com.br

Abstract. *In Software Distributed Development problems such as concurrency and deadlock because of the distributed characteristic affect artifacts of the testing process. In general software projects does not manager them for that distributed teams could access them in any repository without to cause inconsistent retrieval or inconsistent update. A coherent architecture for distributed could contribute for management these artifacts in a testing process. In a distributed development environment based on proposed architecture artifacts such as test scenarios, test cases, stubs, drivers and test results could be right shared.*

Resumo. *No Desenvolvimento Distribuído de Software, problemas oriundos da característica distribuída tais como concorrência e deadlock afetam artefatos do processo de testes. A maioria dos projetos de software não os gerencia de maneira a evitar retornos ou atualizações inconsistentes desses artefatos. Uma arquitetura que considere tais problemas pode contribuir para o gerenciamento desses artefatos no processo de testes. Com o uso de um ambiente de desenvolvimento distribuído baseado em tal arquitetura, cenários de testes, casos de testes, stubs, drivers e resultados de testes podem ser compartilhados adequadamente.*

1. Introdução

No Desenvolvimento Distribuído de Software, as fases de especificação de requisitos, implementação, testes e manutenção necessitam ser modificadas para refletirem a característica distribuída (Schiavoni, 2007).

O processo de testes é uma fase em que muitos artefatos podem ser produzidos e requeridos nas etapas de planejamento, monitoramento, controle e execução do processo de testes tais como cenários de testes, casos de testes, *stubs*, *drivers* de testes e resultados de testes. O gerenciamento desses artefatos não é uma tarefa trivial e se torna mais complexa quando o desenvolvimento de software ocorre entre equipes distribuídas que ocupam localizações físicas diferentes.

Nesse contexto, surgem problemas de um ambiente distribuído em relação aos artefatos de testes tais como a concorrência, *deadlock*s, a maneira como as equipes irão se comunicar, o modo de armazenamento, o compartilhamento, a entrega, dentre outros.

Este trabalho propõe uma arquitetura para o tratamento dos problemas citados, possibilitando um gerenciamento adequado dos artefatos de testes entre equipes distribuídas fisicamente.

Na Seção 2 serão apresentados alguns conceitos empregados neste trabalho. Na Seção 3 serão apresentados alguns trabalhos relacionados encontrados na literatura. Na Seção 4 será apresentada a arquitetura proposta. Na Seção 5 será apresentado um estudo de caso. Finalmente, na Seção 6 serão apresentados as conclusões, limitações e trabalhos futuros.

2. Conceitos Relacionados

Alguns conceitos fundamentais nesse trabalho são artefatos de testes, equipe distribuída, controle de concorrência e *deadlock*. Esses conceitos são descritos nas seções seguintes.

2.1. Artefatos de Testes

Um artefato é o resultado de uma atividade e pode ser utilizado posteriormente como matéria-prima para aquela ou para outra atividade a fim de gerar novos artefatos (Silva e Huzita, 2008).

Segundo McGregor e Sykes (2001), um processo de testes tem o foco voltado para a garantia de que as entradas do software produzam os resultados desejados de acordo com as especificações de requisitos. Desse processo pode resultar uma quantidade enorme de artefatos de testes utilizados ou gerados nas fases de planejamento, monitoramento, controle e execução dos testes. Esses artefatos podem ser cenários de testes, casos de testes, *stubs*, *drivers* de testes e resultados dos testes.

2.2. Equipe Distribuída

Segundo Zanoni (2002), o desenvolvimento distribuído de software, também chamado de desenvolvimento globalizado, caracteriza-se pelo envolvimento de várias equipes no projeto, sendo que pelo menos uma delas está distante fisicamente das demais.

Em alguns casos essas equipes podem ser da mesma organização; em outros casos, podem ser colaborações ou terceirizações (*outsourcing*) envolvendo diferentes organizações. Estas equipes podem estar dentro de um mesmo país ou em países distintos. O desenvolvimento pode ser realizado internamente por uma única organização (*in-house*); numa situação de *outsourcing* a organização contrata outra organização externa para desenvolver um projeto ou parte dele; se, os desenvolvedores contratados ficam alocados no mesmo espaço físico, por exemplo, na sede da organização contratante, formando uma única equipe, denomina-se *nearshore outsourcing* (Carmel e Agarwal, 2001).

No desenvolvimento fisicamente distribuído pode-se ter a existência de centros de desenvolvimento distribuídos, denominados *offshore insourcing*; o envolvimento de subsidiárias; e a participação de outras organizações adquiridas pela organização principal. Pode-se ter o desenvolvimento distribuído envolvendo equipes pertencentes a

organizações independentes, o chamado *offshore outsourcing* (Carmel e Agarwal, 2001).

2.3. Concorrência e *Deadlock*

Nos casos em que há distribuição física das equipes de desenvolvimento de software, localizadas distantes uma da outra, surgem problemas como concorrência e *deadlock*.

Concorrência ocorre quando diversas transações desejam realizar ações de leitura e escrita em dados de maneira intercalada. Devido ao acesso aos mesmos objetos de dados, diversas situações anômalas podem ocorrer caso a intercalação de ações não seja controlada de alguma maneira ordenada. Na teoria de sistemas de banco de dados as situações podem ser de retorno e atualização inconsistentes (Singhal e Shivaratri, 1994).

O retorno inconsistente ocorre quando uma transação lê alguns objetos de dados de um banco de dados antes que outra transação tenha completado com suas modificações naquele objeto de dados. Já a atualização inconsistente de dados ocorre quando muitas transações lêem e escrevem em um conjunto comum de objetos de dados de um banco de dados, deixando-o em um estado inconsistente (Singhal e Shivaratri, 1994).

No processo de testes, um exemplo de concorrência ocorre quando uma equipe A, desenvolvedora de um componente, cria artefatos de testes tais como um conjunto de casos de testes e os disponibiliza para outra equipe B, proprietária de um componente cliente. A equipe B poderá reusar esses casos de testes ao mesmo tempo em que a equipe A modifica esses mesmos artefatos. Com um controle de concorrência adequado, o reuso da equipe B ocorreria somente após as modificações feitas pela equipe A.

Já o d*eadlock* ocorre quando um conjunto de funcionalidades em um sistema é bloqueado devido à espera por requisitos que nunca são satisfeitos. Essas funcionalidades, enquanto satisfazem alguns recursos, requisitam acesso a outros recursos que estão sendo utilizados por outras funcionalidades no mesmo conjunto (Singhal e Shivaratri, 1994).

Existem técnicas tanto para detecção quanto para prevenção de *deadlock*. Na prevenção uma transação é abortada e reiniciada se existe um risco de o *deadlock* ocorrer. Já na detecção, os nós (*sites*) trocam informações a respeito de transações em espera para determinar *deadlocks* globais (Ceri e Pelagatti, 1984).

Um exemplo de ocorrência de *deadlock* no processo de testes é quando um componente, desenvolvido pela equipe E_1, é cliente de algumas das funcionalidades do componente desenvolvido pela equipe E_2 e, ao mesmo tempo, este também é cliente do componente da equipe E_1. Assim, existe uma relação de dependência mútua entre os componentes das equipes.

3. Trabalhos Relacionados

Na literatura foram encontrados alguns trabalhos que propõem arquiteturas específicas para o gerenciamento do processo de testes (Bai et al., 2001; Dias Neto e Travassos, 2006). Outros trabalhos, porém, propõem arquiteturas para o processo de desenvolvimento distribuído (Pascutti, 2002; Silva e Huzita, 2008; Lima e Reis, 2008).

Bai et al. (2001) apresentam uma arquitetura para o gerenciamento de testes completos (*End to End*). Essa arquitetura possibilita realização de testes de sistema e a checagem se este sistema sob teste executa as funções pretendidas do ponto de vista do usuário final. Possibilita gerenciamento de testes remoto bem como a colaboração e cooperação entre *stakeholders* distantes geograficamente. Dias Neto e Travassos (2006) apresentam uma infra-estrutura computacional que apóia o planejamento e controle de testes de software. A ferramenta que implementa essa infra-estrutura denomina-se Maraká e permite o acompanhamento do processo de testes de software e a documentação das atividades realizadas ao longo dos testes usando o padrão internacional IEEE-829.

Pascutti (2002) apresenta uma arquitetura para o DiSEN, um ambiente de desenvolvimento distribuído de software. Essa arquitetura fornece o suporte necessário para o desenvolvimento distribuído de software em que a equipe poderá estar distribuída em locais geográficos distintos e trabalhar de forma cooperativa. Silva e Huzita (2007) apresentam uma melhoria à arquitetura apresentada por Pascutti (2002) com uma estratégia para alocação de artefatos e replicação de repositórios. Lima e Reis (2008) apresentam uma arquitetura para o compartilhamento de informações em ambientes descentralizados de desenvolvimento de software. Essa arquitetura provê apoio a problemas de monitoração e coordenação de processos descentralizados de desenvolvimento de software.

Na Tabela 01 ilustra-se a relação entre esses trabalhos citados e as características da arquitetura proposta. Nota-se que nenhum dos trabalhos citados apresenta todas as características da arquitetura proposta.

Tabela 01 – Relação entre os trabalhos relacionados e as características da arquitetura proposta.

Características	Trabalhos Relacionados					
	Bai et al. (2001)	Dias Neto e Travassos (2006)	Pascutti (2002)	Silva e Huzita (2008)	Lima e Reis (2008)	**Arquitetura Proposta**
Abordagem ao processo de testes	X	X	-	-	-	X
Controle de concorrência	-	-	-	X	-	X
Prevenção de *deadlock*	-	-	X	-	-	X
Comunicação entre as equipes distribuídas	-	-	-	-	-	X
Compartilhamento de artefatos	X	-	X	-	X	X

4. Arquitetura Proposta

A arquitetura proposta neste trabalho pretende alcançar os seguintes objetivos:

- Possibilitar o armazenamento de artefatos de testes em repositórios públicos e privados.
- Possibilitar o acesso aos artefatos armazenados em repositórios de acordo com o modo de acesso (leitura, leitura/escrita) permitido para cada equipe.
- Permitir a comunicação entre as equipes distantes fisicamente.

- Possibilitar o controle de concorrência aos artefatos de testes.
- Possibilitar a prevenção e detecção de *deadlock*.

Na Figura 01 ilustra-se a proposta de arquitetura para atingir tais objetivos. Cada equipe E_i possui dois repositórios: o privado $RPrivE_i$ e o público $RPubE_i$. No repositório privado são armazenados artefatos de testes acessíveis somente à própria equipe enquanto que no repositório público são armazenados artefatos acessíveis às demais equipes. As equipes se comunicam por um canal de comunicação. Já o acesso aos repositórios públicos é realizado por um barramento de dados.

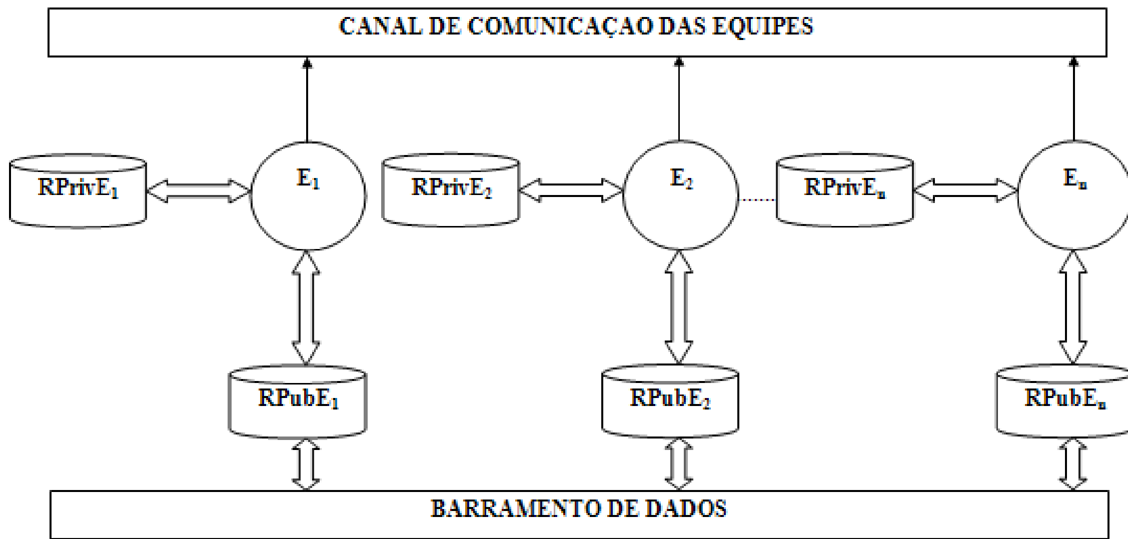

Figura 01 – Arquitetura proposta para o gerenciamento do processo de testes de componentes desenvolvidos por equipes distribuídas.

Os círculos representando as equipes E_1, E_2, ..., E_n na Figura 01 são explodidos para representar as sub-equipes e os módulos gerenciadores na Figura 02. As sub-equipes também têm acesso tanto aos repositórios públicos quanto aos privados. Os módulos gerenciadores tratam dos problemas de concorrência e *deadlock* e do compartilhamento de artefatos de testes.

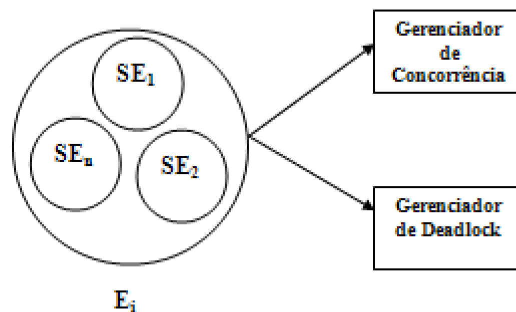

Figura 02 – Módulos gerenciadores de uma equipe E_i composta de sub-equipes SE_1, SE_2,..,SE_n.

Nas seções seguintes será detalhado o barramento de dados, o canal de comunicação entre as equipes e os módulos gerenciadores.

4.1. Barramento de Dados

O barramento de dados é o meio pelo qual os artefatos de testes irão trafegar até chegar ao seu destino.

4.2. Canal de Comunicação entre as Equipes

Na arquitetura proposta neste trabalho, as equipes se comunicarão por meio de trocas de mensagens. O tráfego dessas mensagens ocorrerá em um canal de comunicação.

Uma equipe pode desejar enviar uma mensagem, por exemplo, de que uma das interfaces providas de seu componente sofreu alguma modificação. No entanto, essa equipe quer enviar a mensagem somente para as equipes que dependam desse componente. Uma maneira de fazê-lo é definindo o tipo de entrega da mensagem.

A entrega de mensagens de uma equipe para as demais pode ser de três tipos: *broadcast*, *multicast* e *unicast*, como ilustrado na Figura 03. No tipo *broadcast*, as mensagens são enviadas a todas as demais equipes, por exemplo, E_1 pode enviar mensagens a todas as demais equipes. Já no tipo *multicast*, as mensagens são enviadas a um grupo específico de equipes, por exemplo, E_1 envia mensagens à SE_1, SE_2 e SE_3 de E_3. E no tipo *unicast*, as mensagens são enviadas a uma única equipe específica, por exemplo, E_1 envia mensagens somente à SE_1 de E_3.

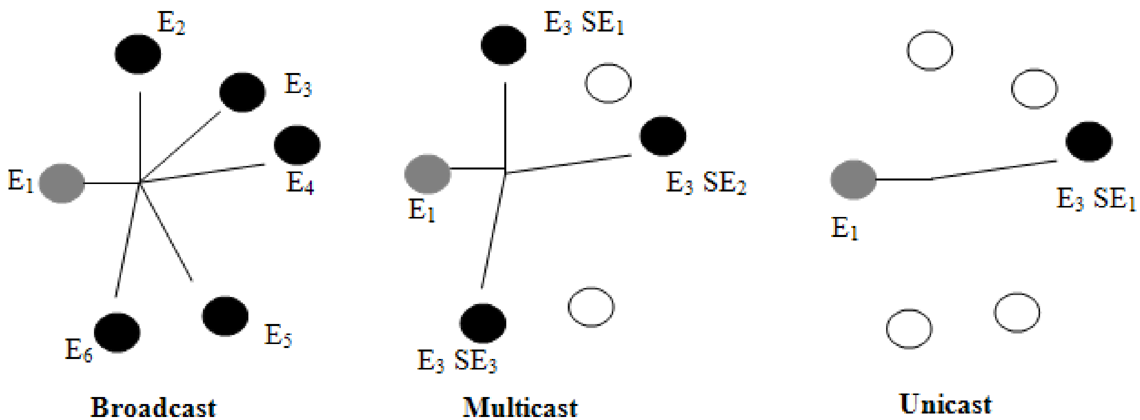

Figura 03 – Tipos de entrega de artefatos de testes: broadcast, multicast ou unicast.

4.3. Repositórios Públicos e Privados

Cada equipe necessita armazenar os artefatos correspondentes aos testes realizados. Esse local de armazenamento é o repositório. O repositório pode ser público (RPub) ou privado (RPriv).

Um repositório público armazena artefatos de testes que podem ser acessados não só pela equipe que os produziu como também por equipes que também reusem esses artefatos tais como cenários de testes de integração, *stubs* de componentes colaboradores etc. Já um repositório privado armazena artefatos que podem ser acessados somente pela equipe que os produziu tais como cenários de testes de unidade, casos de testes de unidade etc.

4.4. Gerenciador de Concorrência

Uma equipe pode acessar um artefato de testes para fazer somente uma leitura (*Read* - R), uma escrita (*Write* - W) ou uma leitura/escrita (R/W) de dados. Quando duas equipes acessam o mesmo artefato de testes com o intuito de fazer somente uma leitura, nenhum problema de concorrência ocorre. No entanto, quando um conjunto de equipes acessa concorrentemente o mesmo artefato de testes e uma delas faz uma operação de escrita, então conflitos podem existir (Ceri e Pelagatti, 1984).

Operações do tipo R(x, a) e W(y, a), em que a transação "x" lê "a" e transação "y" escreve em "a", respectivamente, são encontradas em transações de banco de dados. Fazendo-se uma adaptação dessas operações para o contexto deste trabalho, tem-se:

- $R(E_1, CT_1)$: a equipe E_1 realiza uma operação de leitura no artefato de testes CT_1, que representa um cenário de testes.
- $W(E_2, CT_1)$: a equipe E_2 realiza uma operação de escrita no mesmo artefato de testes CT_1.

Nesse exemplo, se as operações $R(E_1, CT_1)$ e $W(E_2, CT_1)$ ocorrerem concorrentemente, surgirão conflitos uma vez que o cenário de testes CT_1 ao mesmo tempo em que é lido por E_1 também é escrito por E_2.

O gerenciador de concorrência trata das situações de retorno e atualização inconsistentes de artefatos de testes nos repositórios públicos e privados das equipes distribuídas. Esse módulo utiliza um mecanismo que controla a ordem relativa ou intercalada de ações conflitantes, de maneira que toda equipe veja um estado consistente do repositório, e quando todas as equipes já tiverem acessado o repositório, este esteja em um estado consistente. Esse mecanismo é denominado serializabilidade (Bernstein e Goodman, 1981; Papadimitriou, 1979).

4.5. Gerenciador de *Deadlock*

Em relação aos artefatos de testes, *deadlock* pode ocorrer quando, por exemplo, uma equipe E_1 necessita dos testes de integração de um componente da equipe E_2 mas estes ainda não estão prontos e, mutuamente, a equipe E_2 também necessita dos testes de integração do componente da equipe E_1. Nesse caso, existe um ciclo de dependência entre as equipes, que deve ser quebrado pois, caso contrário, ambas equipes não poderão continuar os testes.

Esse módulo gerenciador identifica casos de *deadlocks* como este, de acordo com as relações de dependência dos componentes, e os gerencia de tal forma que os ciclos sejam quebrados. Esse módulo trata tanto da detecção quanto da prevenção de *deadlock* no acesso aos artefatos de testes pelas equipes distribuídas (Ceri e Pelagatti, 1984).

5. Estudo de Caso Baseado no Projeto Tidia-Ae (Tecnologia da Informação no Desenvolvimento da Internet Avançada – Aprendizado Eletrônico)

No estudo de caso apresentado neste trabalho, será utilizado como exemplo o projeto Tidia-Ae (2004), um sistema que auxilia as atividades de aprendizado eletrônico, oferecendo suporte ao aprendizado presencial. Será apresentada uma

adaptação dos núcleos do projeto Tidia-Ae à arquitetura proposta para o gerenciamento do processo de testes.

5.1. O Projeto Tidia-Ae

Para o desenvolvimento do projeto Tidia-Ae, foram criados quatro núcleos de pesquisa principais, distribuídos em algumas universidades do estado de São Paulo, a saber: LARC/EPUSP (São Paulo), LAI/ITA (São José dos Campos), INTERMIDIA/USP (São Carlos) e e-Labora/UNICAMP (Campinas). Cada um desses núcleos é composto de outros sub-núcleos: LARC/EPUSP (Love-me/Uniban - SP, Interlab/EPUSP, CEPA/IF – USP), LAI/ITA (LSI/EPUSP, LaTin/IME – USP), INTERMIDIA/USP (CDCC/USP - São Carlos, LECH/UFSCar, NOMADS/USP - São Carlos, LIEM-GPIMEN/UNESP – Rio Claro, Lince, LISI/USP – Ribeirão Preto), e-Labora/UNICAMP (LAMPA & LACAF/UNICAMP, LED_DIS/Unifesp, LIPACS-UNICAMP, LIA/UFSCar).

Cada núcleo é responsável por um conjunto de funcionalidades tais como gerenciador de contexto, gerenciador de usuários, gerenciador de participantes, gerenciador de ferramentas e ferramentas como chat, fórum de discussão, *whiteboard*, editores de atividades de aprendizagem etc.

O compartilhamento de informações é realizado em um portal (Tidia-Ae, 2004) onde são postados todos os documentos relacionados ao projeto, desde as especificações de projeto ao código implementado. Códigos fonte são disponibilizados somente para os pesquisadores e colaboradores do projeto, que possuem um "nome de usuário/senha" de acesso, mas nenhum controle de acesso é realizado em relação à permissão.

Já a comunicação entre os membros dos núcleos é realizada por meio de ferramentas de bate-papo tais como Skype, MSN, GTalk etc.; reuniões presenciais; troca de emails; telefone; etc.

É dada grande importância às fases de especificação de requisitos e implementação, com a utilização de documentos e tecnologias de desenvolvimento padronizados. No entanto, alguns pontos negativos foram observados, a saber:

- O modo de compartilhamento de informações não permite o controle de concorrência e *deadlock* dos dados.
- A maneira de comunicação entre os membros dos núcleos depende de ferramentas externas ao ambiente de desenvolvimento.
- Artefatos de testes do projeto tais como testes de unidade, de integração, de sistema, de regressão, dentre outros, não possuem um caráter tão relevante quanto o necessário.

5.2. Aplicação da Arquitetura Proposta ao Projeto Tidia-Ae

Em relação às equipes distribuídas, cada núcleo do projeto Tidia-Ae pode ser considerado uma equipe E_i e cada sub-núcleo pode representar uma sub-equipe SE_j dentro da equipe. As sub-equipes ficam subordinadas à equipe e utilizando o seu repositório. Como exemplo, E_2 LAI/ITA tem como sub-equipes SE_1 LSI/EPUSP e SE_2 LaTin/IME-USP. E_2, SE_1 e SE_2 utilizam os mesmos repositórios público $RPubE_2$ e

privado $RPrivE_2$. Também possuem os mesmos privilégios de acesso aos artefatos, uma vez que fazem parte da mesma equipe E_2. No entanto, E_2 LAI/ITA, SE_1 LSI/EPUSP e SE_2 LaTin/IME-USP só poderão acessar repositórios públicos de equipes remotas que tenham alguma dependência de funcionalidade.

No que se refere ao acesso a repositórios, pode-se citar a dependência de funcionalidade da ferramenta “Fórum de Discussão”, desenvolvida por E_2 - LAI/ITA, dependente das ferramentas “Gerenciador de Participantes” e “Gerenciador de Ferramentas”, ambas desenvolvidas por E_1 - LARC/EPUSP. Nesse caso de dependência funcional, E_1 - LARC/EPUSP permitirá que E_2 - LAI/ITA acesse o seu repositório público em busca de artefatos de testes tais como cenários de testes que auxiliem na realização de testes de integração, por exemplo. No entanto, cenários de testes de unidade de E_1 - LARC/EPUSP não são importantes para E_2 - LAI/ITA e então estes poderão ser armazenados em seu repositório privado.

Ainda, E_1 - LARC/EPUSP não permitirá que E_2 - LAI/ITA acesse cenários de testes de integração quando estes estiverem sendo modificados. A liberação de acesso só ocorre quando as modificações são finalizadas.

Em relação à ocorrência de *deadlock*, um exemplo seria a relação de dependência mútua entre o módulo “Gerenciador de Ferramentas” e “Fórum de Discussão”, desenvolvidos por E_1 - LARC/EPUSP e E_2 - LAI/ITA, respectivamente. Há a ocorrência de *deadlock* quando ambas as equipes decidem realizar testes de integração simultaneamente. Esse *deadlock* pode ser resolvido com a quebra do ciclo de dependência pelo módulo Gerenciador de *Deadlock* proposto na arquitetura. Este módulo informará, por exemplo, que E_2 - LAI/ITA deve disponibilizar um *stub* de algumas de suas interfaces para que E_1 - LARC/EPUSP possa realizar os testes de integração.

6. Conclusões, Limitações e Trabalhos Futuros

Problemas oriundos da característica de distribuição, como concorrência e *deadlock*, aumentam a complexidade das atividades de desenvolvimento de software, especificamente o gerenciamento de artefatos do processo de testes tais como cenários de testes, casos de testes, *stubs*, *drivers* de testes e resultados de testes.

A arquitetura apresentada neste trabalho propõe tal gerenciamento. Módulos gerenciadores tratam os problemas de concorrência e controle, um canal de comunicação permite a troca de mensagens entre as equipes e um barramento de dados possibilita o tráfego de artefatos armazenados em repositórios.

Uma das limitações da arquitetura proposta é a quantidade de equipes suportada. Essa limitação implica no projeto de um canal de comunicação e de um barramento de dados que suporte o tráfego de mensagens correspondente à quantidade de equipes. A capacidade de armazenamento dos repositórios de artefatos também deve ser estimada de acordo com a capacidade de desenvolvimento de artefatos da equipe. Uma ferramenta deve implementar essa arquitetura.

A implementação de um protótipo de ferramenta baseado na arquitetura proposta e especificação de atividades do processo de testes para a produção de artefatos são alguns dos trabalhos futuros.

Referências

Bai, X., Tsai, W. T., Shen, T., Li, B., Paul, R. (2001). Distributed End-to-End Testing Management. Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing. IEEE Computer Society, Washington, DC, p. 140.

Bernstein, P. e Goodman, N. (1981) Concurrency Control in Distributed Database Systems. ACM Computing Surveys.

Carmel, E. e Agarwal, R. (2001) Tactical Approaches for Alleviating Distance in Global Software Development. IEEE Software, p. 22-29.

Ceri, S. e Pelagatti (1984) Distributed Databases Principles and Systems. McGraw-Hill, USA.

Dias Neto, A.C., Travassos, G.H. (2006). Maraká: Infra-estrutura Computacional para Apoiar o Planejamento e Controle de Testes de Software. V Simpósio Brasileiro de Qualidade de Software (SBQS). Vila Velha, ES.

Lima, A. M. e Reis, R. Q. (2008) Compartilhamento de Informações sobre Processos em Ambientes Descentralizados de Desenvolvimento de Software. III Workshop de Desenvolvimento Distribuído de Software – WDDS. Campinas – SP.

McGregor, J. D., Sykes, D. A. (2001). A Practical Guide to Testing Object-Oriented Software. Addison-Wesley, New Jersey.

Papadimitriou, C.H. (1979) Serializability of Concurrent Updates. Journal of the ACM.

Pascutti, M. C. D. (2002) Uma Proposta de Arquitetura de um Ambiente de Desenvolvimento de Software Distribuído Baseada em Agentes. Dissertação de Mestrado. UFRGS. Porto Alegre – RS.

Schiavoni, F. L. (2007) FRADE – Framework para Infra-Estrutura de um Ambiente Distribuído de Desenvolvimento de Software. Dissertação de Mestrado. UEM – Maringá – Paraná.

Silva, C. A. e Huzita, E. H. M. (2008) DiSEN-SCV: Uma Estratégia para Replicação de Repositórios e Alocação de Artefatos. III Workshop de Desenvolvimento Distribuído de Software – WDDS. Campinas – SP.

Singhal, M. e Shivaratri, N. G. (1994) Advanced Concepts in Operating Systems: Distributed, Database and Multiprocessor Operating Systems. McGraw-Hill, USA.

Tidia-Ae (2004). Disponível no site: http://tidia-ae.usp.br/portal [acessado em 04-julho-2009].

Zanoni, R. (2002) Modelo de Gerência de Projeto Baseado no PMI para Ambiente de Desenvolvimento de Software Fisicamente Distribuído. Dissertação de Mestrado - PUC-RS, Porto Alegre.