CAPÍTULO 8

Desenvolvimento colaborativo de software

Cleidson Ronald Botelho de Souza
Sabrina Marczak
Rafael Prikladnicki

META

Apresentar as características que tornam o desenvolvimento de software uma atividade colaborativa, bem como apresentar sistemas que apoiam as atividades colaborativas durante o ciclo de desenvolvimento de software.

OBJETIVOS EDUCACIONAIS

Após o estudo desse capítulo, você deverá ser capaz de:

- Identificar aspectos colaborativos nas atividades de desenvolvimento de software.
- Analisar a qualidade do apoio à colaboração oferecido por sistemas de desenvolvimento de software tradicionais e modernos.
- Identificar requisitos de colaboração para apoiar o desenvolvimento colaborativo de software.

RESUMO

Neste capítulo é discutido o desenvolvimento de software enquanto atividade colaborativa. É apresentada uma visão geral sobre desenvolvimento de software e como a Engenharia de Software dá suporte à colaboração. São discutidas práticas colaborativas realizadas durante o ciclo de desenvolvimento de um software. São apresentados sistemas computacionais que promovem e apoiam a colaboração durante o desenvolvimento de software, e são comparadas as características dos sistemas tradicionais com as dos sistemas modernos que apoiam a colaboração. Também é discutido como as distâncias física, temporal e cultural afetam a realização de práticas colaborativas na atividade de desenvolvimento de software em ambientes geograficamente distribuídos.

8.1 Desenvolvimento de software como uma atividade colaborativa

O software tornou-se vital para os negócios de todos os tipos de organizações. O sucesso de uma organização depende cada vez mais da utilização de software como um diferencial competitivo. O desenvolvimento de software passou a ser uma atividade essencial no mundo atual. O aumento do uso de sistemas em contextos mais abrangentes de negócio torna a construção de software uma atividade cada vez mais complexa. Diversos especialistas precisam trabalhar em conjunto para desenvolver o software com sucesso. O sucesso depende da entrega dentro do prazo e do custo estimados e com a qualidade desejada pelo cliente. Raramente o desenvolvimento de software é uma atividade individual. Em geral, é uma atividade colaborativa com a atuação de diversos profissionais para projetar soluções e produzir código de qualidade. Os membros de uma equipe de desenvolvimento de software precisam coordenar suas atividades, planejar novas ações, tomar decisões, realizar as atividades previstas e também se comunicar para desenvolver um software. A coordenação envolve programadores e uma série de outros profissionais, como analistas de requisitos, gerentes, arquitetos de software, membros da equipe de garantia de qualidade, projetistas de interface gráficas e assim por diante. Os desenvolvedores de software também precisam alinhar suas atividades com os profissionais de outras áreas como marketing, vendas, finanças, entre outras.

Uma abordagem para facilitar o desenvolvimento de software é a construção colaborativa dos modelos da Engenharia de Software. Modelos, ou artefatos, servem como abstrações do software que está sendo construído. Exemplos de modelos incluem especificações de requisitos, diagramas de casos de uso, diagramas de classe, até efetivamente o código-fonte do software em uma ou mais linguagens de programação. Os modelos variam de acordo com o grau de formalismo: formal, como os programas escritos em linguagens de programação; semiformal, como diagramas; ou informal, como a linguagem natural utilizada nas especificações de requisitos. Cada modelo é utilizado em uma ou mais fases do processo de desenvolvimento de software (análise, projeto, codificação, testes e implantação), e é construído e mantido por profissionais que desempenham diferentes papéis como programadores, analistas e gerentes. A colaboração possibilita que vários engenheiros de software façam contribuições de maneira coordenada e sem conflitos a um ou mais modelos.

Os modelos não são independentes uns dos outros. Os modelos se relacionam de tal forma que mudanças em um modelo em geral implicam em mudanças nos modelos associados. Por exemplo, quando modificações são feitas no diagrama de classes de um software, são necessárias modificações no código-fonte para que os dois modelos permaneçam consistentes entre si. A dependência torna a colaboração e a coordenação entre os engenheiros de software ainda mais importante, pois se as mudanças não forem alinhadas corretamente, os modelos se tornarão diferentes, o que pode levar a erros.

Existem vários sistemas, tanto comerciais quanto acadêmicos, que dão suporte à colaboração entre os profissionais através da construção colaborativa de modelos. Estes sistemas vão desde a especificação colaborativa de requisitos até a construção de diagramas de maneira síncrona ou assíncrona, passando por editores de texto cooperativos que possibilitam que diferentes programadores escrevam o código ao mesmo tempo, até sistemas que gerenciam o acesso compartilhado a artefatos.

ENGENHARIA DE SOFTWARE E SUAS CAMADAS

A atividade de desenvolvimento de software é regida pela disciplina da Engenharia de Software. Nesta disciplina são fornecidos mecanismos, sistemas e princípios para que os profissionais desenvolvam software de qualidade no custo e no prazo determinados. Pressman (2006) entende a Engenharia de Software através de camadas – Figura 8.1. As camadas abrangem três elementos fundamentais: ferramentas, métodos e processo. Cada um dos elementos corresponde a uma camada, sendo que a camada base representa o foco na qualidade. Esta organização implica que as camadas que representam os elementos fundamentais devem possibilitar à equipe o controle do processo de desenvolvimento e oferecer uma base para a construção de software de alta qualidade.

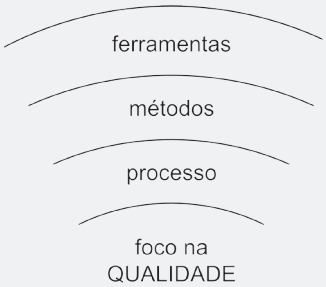


Figura 8.1 Camadas da Engenharia de Software

Fonte: Pressman, 2006, p.17

Os métodos proporcionam os detalhes de "como fazer" para desenvolver o software. Envolvem um amplo conjunto de tarefas que incluem: planejamento e estimativa de projeto, análise de requisitos, projeto da estrutura de dados, arquitetura do software e algoritmo de processamento, codificação, teste, manutenção, entre outras. Ferramentas diversas dão apoio a diferentes métodos.

Um processo possibilita o desenvolvimento racional do software. Define a sequência em que os métodos serão aplicados, como os produtos serão entregues, os controles que ajudam a assegurar a qualidade e a coordenar as mudanças, e os marcos de referência que possibilitam à equipe avaliar o progresso do desenvolvimento. É representado por um modelo de processo operacionalizado por meio de uma metodologia. Existem diversos modelos de processo de desenvolvimento de software, e cada modelo de processo pode ter mais do que uma metodologia que o operacionaliza. A metodologia estabelece a sequência das atividades e o momento em que os métodos e as ferramentas são utilizados.

8.2 Práticas colaborativas no desenvolvimento de software

Algumas práticas da Engenharia de Software facilitam a colaboração durante o desenvolvimento de um software. Um exemplo é o próprio processo de software, pois requer a atuação de vários profissionais especializados. Outra prática é a programação em pares empregada em métodos ágeis.

8.2.1 Processo de software

No contexto da Engenharia de Software, um processo é um conjunto de métodos, práticas e transformações que um grupo de pessoas emprega para desenvolver e manter software e os produtos ou modelos. Exemplos de produtos de software são: planos de projeto,

documentos de projeto, código, casos de teste, manual do usuário. Um processo de software define a sequência em que os métodos serão aplicados, como os produtos serão entregues, os controles que ajudam a assegurar a qualidade e a coordenar as mudanças, e os marcos de referência que possibilitam aos gerentes avaliar o progresso do desenvolvimento do software.

Uma forma de facilitar a atividade de desenvolvimento do software durante o ciclo de vida é por meio da definição de papéis ou funções em processos de software. Exemplos de papéis incluem programadores, gerentes, analistas de requisitos e arquitetos de software. Um papel descreve como um indivíduo se relaciona com o modelo ou artefato compartilhado e com os outros indivíduos do grupo. A cada papel é atribuído um conjunto de operações que podem ser executadas. Por exemplo, membros da equipe de qualidade de software são responsáveis por identificar defeitos no código-fonte, enquanto os programadores são responsáveis por consertar os defeitos identificados. A atribuição de papéis é um mecanismo para facilitar a coordenação das atividades dos vários atores envolvidos com os modelos. Num processo de software estão definidos que papéis serão exercidos e quando irão atuar no desenvolvimento de um software.

ESTUDOS PRECURSORES DA VISÃO COLABORATIVA DE DESENVOLVIMENTO DE SOFTWARE

A Engenharia de Software reconhece que desenvolvimento de software é uma atividade colaborativa. Por exemplo, em um estudo clássico, Curtis e colegas identificaram "problemas de comunicação e coordenação de atividades" para o desenvolvimento de software. De maneira similar, vários estudos sugerem que a maior parte do tempo gasto pelos engenheiros de software é em atividades colaborativas e não em atividades individuais. O precursor destes resultados é o trabalho de Perry e coautores que indica que engenheiros de software passam até 50% do tempo envolvidos em comunicação formal e informal com os colegas. O grande volume de comunicação ocorre tanto em empresas norte-americanas quanto brasileiras (Gonçalves 2009). Os estudos citados sugerem que o trabalho diário de um engenheiro de software envolve um porcentual tão grande de atividades colaborativas quanto de atividades individuais.

O processo de software impõe um fluxo de colaboração entre pessoas que desempenham papéis na execução das atividades previstas no processo. A definição de papéis e atividades para guiar o desenvolvimento de software facilita a coordenação das atividades colaborativas, pois possibilita que os atores envolvidos entendam o contexto de desenvolvimento e como suas atividades, e as atividades dos colegas, afetam uns aos outros. A definição das atividades do processo, que definem quando cada modelo vai ser construído, também possibilita a identificação das dependências entre os modelos.

8.2.2 Programação em pares

Programação em pares é uma prática proposta no método ágil conhecido como Extreme Programming, em que dois programadores atuam juntos em um único computador. Em geral, a dupla é formada por um profissional iniciante e outro mais experiente de forma que

o iniciante fica à frente codificando enquanto o mais experiente acompanha a codificação e apoia o desenvolvimento das habilidades do colega. Com o uso desta prática de programação em pares, o código em desenvolvimento sempre é revisado em tempo real por duas pessoas, o que diminui a possibilidade de defeitos e potencializa a melhora da qualidade do código-fonte gerado, ao mesmo tempo em que promove a constante evolução da equipe. Se por um lado o profissional iniciante fará muitas perguntas e digitará muito pouco, o profissional mais experiente identificará pequenos erros do iniciante. Em pouco tempo o profissional iniciante tem condições de compreender como o profissional experiente trabalha e perceber os possíveis erros que comete e, assim, melhorar a qualidade do trabalho que desenvolve.

Na programação em pares, enquanto um integrante da dupla está pensando na melhor forma de implementar um requisito, o outro tenta responder as seguintes perguntas: Essa abordagem vai funcionar? Existem outros casos de teste que podem não funcionar? Existe alguma forma de simplificar o sistema?

A programação em pares requer uma mudança cultural. A realização da prática não consiste em uma pessoa programando enquanto outra assiste. A programação em pares é um diálogo entre duas pessoas que programam de forma simultânea e que buscam programar melhor do que quando atuam isoladamente. Além de programar, em geral, duas pessoas também analisam, projetam e testam os códigos. A tendência é de que em alguns meses diminua a diferença entre os profissionais, ao mesmo tempo em que são identificados pontos fortes e fracos de cada um. Como consequência da evolução dos profissionais, espera-se o aumento da produtividade, qualidade e satisfação das pessoas envolvidas nesta prática.

Para que a programação em pares funcione bem, é necessário investir algum tempo planejando os pares para evitar conflitos desnecessários. Segundo Kent Beck, um dos criadores da Extreme Programming, a programação em pares é mais produtiva do que dividir o trabalho entre dois programadores e então integrar os resultados. Selecionar os pares é um ponto delicado dentro das equipes de desenvolvimento quando não existe maturidade suficiente para compartilhar o código que está sendo desenvolvido. Uma sugestão para equipes iniciantes é desenvolver uma iteração onde se faz todo o código em pares e outra iteração onde o código é feito individualmente. A partir do resultado, em cada projeto de desenvolvimento de software deverá se tomar a decisão de como alocar os pares.

A programação em pares exemplifica uma prática para a construção colaborativa de modelos, neste caso, o código-fonte. A prática de programação em pares é um exemplo de atividade social e colaborativa entre dois integrantes de uma equipe. A prática envolve diversas atividades técnicas, como teste e refatoração do código, e também envolve atividades sociais como a intensa comunicação entre a dupla envolvida na atividade. Compartilhamento de visões, investigação e entendimento de razões para escolher determinado caminho e negociação são aspectos da colaboração presentes na programação em pares. Esta prática contribui para melhorar o processo de desenvolvimento de software como um todo, pois os envolvidos aprendem a enxergar o software sob diferentes perspectivas seguindo o viés de colaboração e melhoria contínua.

8.3 Sistemas colaborativos tradicionais de desenvolvimento

O objetivo desta seção é apresentar e exemplificar os dois tipos de sistemas colaborativos mais utilizados em atividades de desenvolvimento de software: sistemas de gerência de configuração e sistemas de gerência de defeitos (bugs). Estes sistemas são ditos tradicionais porque são utilizados em projetos de software de forma extensa durante o ciclo de desenvolvimento de software.

8.3.1 Sistemas de controle de versão

Gerência de configuração é a disciplina responsável por controlar a evolução e a integridade de um produto de software por meio da identificação da configuração do sistema em diferentes momentos do ciclo de desenvolvimento. Para apoiar o desenvolvimento de software através do controle e registro das mudanças, as principais funções exercidas pela gerência de configuração são: identificação da configuração, controle da configuração, registro dos estados da configuração, avaliações e revisões da configuração, e gerenciamento da entrega ou versões do software.

Para manter a integridade de um produto de software, é necessário manter a consistência entre os itens que compõem o produto de software. Para garantir a consistência, sistemas que implementam as funções de gerência de configuração são adotados em larga escala por equipes de desenvolvimento de software. Estes sistemas são chamados de sistemas de controle de versão, e os exemplos mais conhecidos são o CVS e o Subversion.

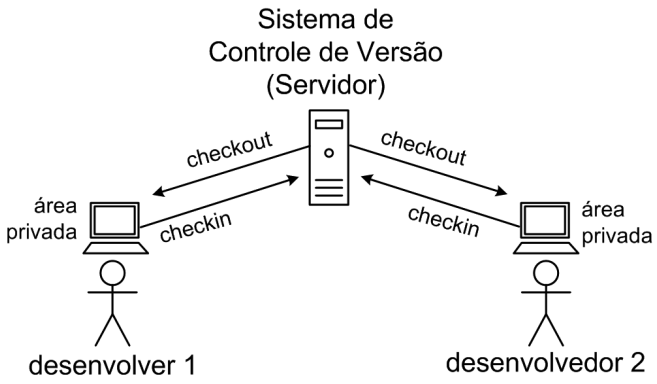


Figura 8.2 Compartilhamento de arquivos em um sistema de controle de versão

Os atuais sistemas de gerência de configuração apoiam a coordenação necessária para manter a integridade dos produtos de software da seguinte forma:

- Geram relatórios de mudanças, de progresso e de composição de itens de um produto e apoiam a comunicação entre os desenvolvedores.
- Fornecem informação de percepção sobre o que os colegas desenvolvedores estão fazendo, notificam mudanças e outros eventos ocorridos em qualquer um dos itens de configuração atrelados ao sistema.

- Possibilitam a coordenação de dependências entre as atividades, seja para apoiar a política de desenvolvimento sequencial ou em paralelo com junção do material desenvolvido por cada indivíduo em momentos predefinidos.
- Oferecem um espaço para os desenvolvedores compartilharem o artefato produzido quando concluído, o que evita a notificação das alterações parciais que não afetam o desenvolvimento das atividades dos colegas. O arquivo é armazenado em um repositório central, no qual todos enviam (check-in) seus arquivos e individualmente podem copiar (check-out) o arquivo para uma área privada para fazer alterações antes de enviar novamente para o repositório central e permitir que os demais integrantes da equipe tenham acesso ao arquivo (Figura 8.2).
- Atuam como uma memória compartilhada do histórico das atividades do produto em desenvolvimento por meio de métricas estatísticas ou relatórios sobre os itens de configuração, o que possibilita a visualização da evolução do sistema.

8.3.2 Sistemas de gestão de defeitos (Bugs)

Um software raramente está livre de defeitos. Em geral, defeitos são encontrados pela equipe de qualidade de software ou pelos usuários finais. Identificar e remover os defeitos é importante para a qualidade final do software. Existem várias técnicas para identificar defeitos, tais como inspeções e revisões por pares de código-fonte, teste unitário e teste de usuário. Estas técnicas requerem atividades colaborativas entre desenvolvedores, ou entre desenvolvedores e usuários do sistema. Por exemplo, em revisão por pares, os desenvolvedores revisam em conjunto o código-fonte para detectar defeitos no código. Sistemas colaborativos denominados sistemas de gerência de defeitos, como o Bugzilla e o JIRA, possibilitam o gerenciamento da solução dos defeitos.

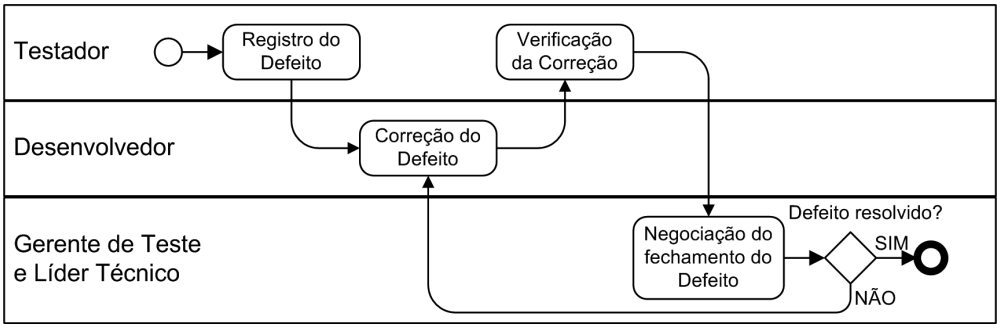


Figura 8.3 Colaboração entre os atores que atuam no ciclo de vida de um defeito de software

Os sistemas de gerenciamento de defeitos possibilitam o registro de um relatório inicial sobre o defeito identificado, o qual após ser analisado por um engenheiro de software, é encaminhado a um desenvolvedor para que trabalhe na solução do defeito descrito. Uma vez que o defeito tenha sido solucionado, uma descrição da solução é adicionada ao sistema para que a equipe de teste possa verificar se o defeito foi de fato removido, e se as alterações no código-fonte não acrescentaram erros ou causaram outros defeitos. O testador responsável, por sua vez, pode rejeitar a correção do desenvolvedor caso identifique que o defeito não

foi corrigido de forma satisfatória. Usuários finais também têm acesso ao sistema e opinam sobre a remoção dos defeitos que haviam identificados. Documentos para dar apoio e facilitar a compreensão da descrição do defeito são acrescentados ao sistema em qualquer estágio do ciclo de vida de um defeito. Um exemplo dos envolvidos neste ciclo de vida é apresentado na Figura 8.3. A maioria destes sistemas possibilita que todos os atores se comuniquem por troca de mensagens textuais. As mensagens ficam associadas aos defeitos, pois muitas vezes documentam parte do processo que levou a resolução.

Além do registro e solução do defeito, os sistemas de gerenciamento de defeitos também possibilitam a priorização dos defeitos a serem corrigidos pelo desenvolvedor, a geração de relatórios de progresso e do estado atual de cada defeito registrado no sistema, a identificação de registro de defeitos similares ou mesmo repetidos, e a busca por determinados defeitos. Possibilitam ainda que diferentes atores envolvidos no desenvolvimento de software possam coordenar as atividades por meio do compartilhamento de informações que são relevantes em um determinado momento.

Estes sistemas são em geral integrados a ambientes de desenvolvimento de software e de especificação de requisitos, bem como de gerenciamento de projeto para que os gerentes possam acompanhar diretamente a qualidade do sistema pelos quais são responsáveis. Equipes geograficamente distribuídas também se beneficiam diretamente deste tipo de sistema, pois seus integrantes, na maioria das vezes, não têm a oportunidade de se encontrarem pessoalmente para detectar defeitos em conjunto ou para discutir como solucioná-los. Versões destes sistemas para web são cada vez mais comuns.

8.4 Sistemas colaborativos atuais de desenvolvimento de software

Nesta seção são apresentados dois sistemas modernos usados em atividades de desenvolvimento de software e que fornecem apoio a atividades colaborativas. O objetivo é discutir a tendência de integração de sistemas para suporte ao desenvolvimento de software, e destacar as funcionalidades para atividades colaborativas no desenvolvimento de software, como por exemplo, bate-papo integrado ao código-fonte, marcadores (tags) e notificações de progresso e de alterações. Outro objetivo desta seção é ilustrar como os conceitos da área de Sistemas Colaborativos são aplicados em sistemas de desenvolvimento de software para facilitar a colaboração. Apesar de ter sido objeto de estudo desde os primeiros anos dessa área, apenas recentemente as lições aprendidas em Sistemas Colaborativos começaram a influenciar o projeto de sistemas para o desenvolvimento de software.

8.4.1 IBM Rational Team Concert

IBM Rational Team Concert é um sistema colaborativo de desenvolvimento de software construído a partir da arquitetura denominada Jazz, também da IBM. É composto de um ambiente integrado de desenvolvimento de software, de um sistema de controle de código-fonte, de um sistema de gerenciamento de erros e defeitos, e com a integração de sistemas colaborativos como bate-papo e mecanismos de percepção do contexto do trabalho em grupo.

Este ambiente integrado oferece apoio a diversos aspectos do processo de desenvolvimento de software, incluindo planejamento ágil, definição de processos, controle de alteração de

código-fonte, registro e gerenciamento de erros e defeitos, gerenciamento de entregas (builds), e relatórios gerenciais e técnicos. O sistema é usado para gerenciar relações e dependências entre artefatos de software, promover práticas de desenvolvimento e obter informações sobre o projeto em andamento.

Os usuários do Rational Team Concert criam itens de trabalho, como por exemplo requisitos, atividades, defeitos, para acompanhar o desenvolvimento das atividades do projeto. Entre outras funcionalidades, os itens de trabalho são associados a datas de entrega definidas no plano de projeto ou a certos arquivos de código-fonte. Quando um item de trabalho é criado ou modificado, todos os outros indivíduos associados ao item são notificados, o que promove a coordenação entre os membros da equipe por meio da percepção das atividades dos colegas e, assim, apoia a colaboração.

Outra característica que promove colaboração é a comunicação integrada aos itens de trabalho. Quando dois desenvolvedores se comunicam usando o sistema, as conversas relacionadas ao item de trabalho são registradas e automaticamente disponibilizadas para os demais que atuam naquele item de trabalho. Com este bate-papo integrado, o uso de correio eletrônico ou qualquer outro sistema de comunicação não embutido no Rational Team Concert é desencorajado para que se maximize os benefícios do sistema.

Outra característica de destaque é o suporte a um processo iterativo de software. Este suporte se dá por meio da funcionalidade de geração de entregas contínuas e com acesso ao sistema de gerenciamento de controle de código-fonte, o que minimiza a postergação de identificação de erros em etapas futuras e permite a coordenação de atividades entre diversos colaboradores do escopo de projeto sendo entregue.

8.4.2 Microsoft Visual Studio Team Foundation Server

Microsoft Visual Studio Team Foundation Server é uma plataforma para o gerenciamento do ciclo de vida de aplicações que oferece suporte para equipes que trabalham especificamente com a tecnologia .NET e outras tecnologias como Java ou Cobol. Possui um conjunto de características para dar suporte a colaboração entre integrantes de uma equipe de software, tais como, integração de sistemas de mensagem instantâneas e notificação de alterações em registros. O sistema possibilita interações entre os desenvolvedores e testadores de uma equipe, o que facilita a colaboração entre os atores que atuam nestes dois papéis.

Testadores trabalham para encontrar defeitos em uma aplicação, enquanto desenvolvedores trabalham para adicionar funcionalidades, modernizá-las, e também para corrigir defeitos encontrados. Recursos e mecanismos foram criados para promover mais sinergia e facilitar a interação entre estes dois papéis. O sistema dá mais visibilidade da contribuição de ambos num projeto, o que promove mais colaboração. Recursos como a possibilidade de registrar os defeitos identificados com informações detalhadas, a rápida busca por informações para a descoberta da causa de um defeito e a automatização da execução dos testes são alguns exemplos que fazem do Team Foundation Server uma plataforma que apoia a colaboração entre testadores e desenvolvedores.

A possibilidade de se registrar defeitos com informações detalhadas consiste na captura automática de informações do ambiente onde os testes são realizados, como por exemplo, todos

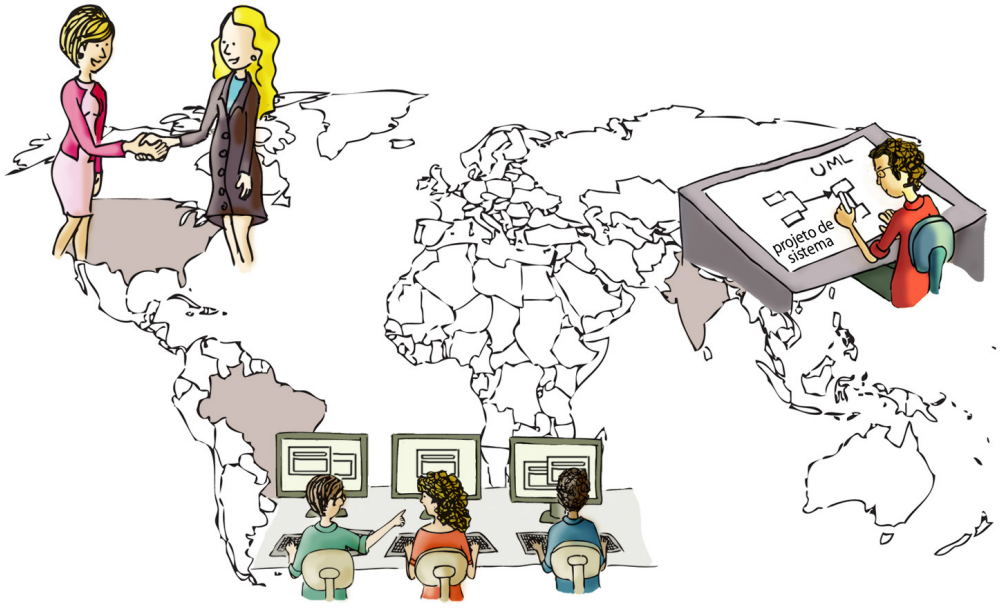
os passos executados por um testador incluindo o tempo gasto em cada passo, imagens ou mesmo um vídeo das ações executadas. Quando um testador relata um defeito num dado caso de teste, o defeito chega até a equipe de desenvolvimento com uma boa quantidade de informação, capturada automaticamente, o que permite um desenvolvedor identificar o lugar no código onde um defeito ocorreu. Parte deste conjunto de informações atua como um debug histórico da aplicação, o que evita que defeitos sejam encerrados por "não serem reproduzidos" ou por "não existirem". Os desenvolvedores interagem com os testadores para esclarecer informações e corrigir os defeitos relatados visando diminuir as pendências em relação ao processo de teste.

O sistema também possibilita testadores e desenvolvedores trabalharem de forma mais colaborativa e integrada na alteração do código-fonte. São dados alertas dos testes que precisam ser executados novamente devido a uma alteração no código-fonte. Esta funcionalidade possibilita que testadores identifiquem quais testes precisam ser executados antes da entrega a fim de garantir que um defeito tenha sido corrigido.

A equipe cria conteúdos colaborativamente em formato de texto por meio de wiki pages ao usar outro sistema integrado, SharePoint, também da Microsoft. Este conjunto de características do sistema promove visibilidade do progresso das atividades e facilita a colaboração entre testadores e desenvolvedores.

8.5 Desenvolvimento distribuído e global de software

Nos últimos anos ocorreu a globalização dos negócios, o que também se refletiu na área de desenvolvimento de software. O mercado global cria novas formas de colaboração e de competição que vão além das fronteiras dos países. Torna-se cada vez mais custoso desenvolver software no mesmo espaço físico, na mesma organização ou até no mesmo país. O avanço da economia, a sofisticação dos meios de comunicação e a pressão por baixos custos tem incentivado o investimento crescente no desenvolvimento distribuído de software.



O desenvolvimento distribuído de software é caracterizado pela distância física e temporal entre os participantes, por exemplo, entre cliente, projetistas, programadores e usuários. O desenvolvimento global de software é caracterizado quando os participantes estão distribuídos em mais de um país (Global Software Development ou Global Software Engineering). Um projeto distribuído e global é executado por equipes de diferentes nacionalidades que trabalham unidas em um projeto comum, embora em culturas e fusos horários distintos.

O desenvolvimento distribuído de software tem sido caracterizado pela colaboração entre departamentos de organizações e pela criação de grupos de desenvolvedores que trabalham em conjunto, mas localizados em cidades ou países diferentes, distantes temporal e fisicamente. Apesar de muitas vezes a distribuição ocorrer em um mesmo país, em regiões com incentivos fiscais ou de concentração de massa crítica em determinadas áreas, algumas empresas buscam soluções globais em outros países visando mais vantagens competitivas, o que potencializa os problemas e os desafios existentes.

O EFEITO DA DISTÂNCIA EM ATIVIDADES COLABORATIVAS

Em 1977, Thomas J. Allen (1977) conduziu um estudo e observou que a frequência de colaboração entre engenheiros era inversamente proporcional à distância física entre as salas: quanto maior a distância física, menor a frequência com que os engenheiros colaboravam. O resultado deste estudo foi mais tarde observado em outras organizações, o que sugere que a distância tem um efeito negativo sobre a colaboração. Mais precisamente, se a distância for maior do que 30 metros, a colaboração entre os atores fica prejudicada.

O efeito é consequência das oportunidades de comunicação informal entre as pessoas. A comunicação formal é a atividade de comunicação predefinida, acordada entre as partes, como por exemplo, uma reunião agendada entre diversos participantes. Por outro lado, a comunicação informal é espontânea, ocorre sem planejamento, por exemplo, quando dois profissionais se encontram durante o "cafezinho" no meio do expediente e começam a conversar sobre o trabalho.

Conversas informais são relevantes para a coordenação de atividades. Profissionais tomam ciência sobre o andamento do trabalho dos colegas (por exemplo, se eles vão terminar suas atividades no prazo ou não), informam os colegas sobre suas próprias atividades, discutem problemas comuns, e até mesmo encontram soluções para vários problemas. A possibilidade das pessoas se engajarem em conversas informais diminui com a distância física e, consequentemente, diminuem as oportunidades para a coordenação informal das atividades. Apesar de vários sistemas colaborativos auxiliarem a reduzir a distância entre as pessoas, como os sistemas de mensagens instantâneas e videoconferência, a distância física entre profissionais que precisam colaborar ainda é um problema.

O desenvolvimento de software tradicional já possui diversas dificuldades. Quando realizado de forma distribuída, são acrescidas ainda dificuldades decorrentes da dispersão geográfica entre os participantes do projeto, da dispersão temporal (diferença de fuso-horário) e das diferenças culturais incluindo idioma, tradições, costumes, normas e comportamento. As di-

ferenças se refletem em diversos fatores: questões estratégicas (decisão de desenvolver ou não um projeto de forma distribuída, tendo por base análises de risco e custo-benefício); questões culturais (valores e princípios entre as equipes distribuídas); questões de infraestrutura (redes de comunicação de dados, plataformas de hardware, ambiente de software); conhecimento técnico (como o processo de desenvolvimento de projetos distribuídos); e questões de gestão do conhecimento (fatores relativos à criação, armazenamento, processamento e compartilhamento de informações nos projetos distribuídos).

Apesar de a distância física ser reconhecida como problemática para atividades colaborativas, estudos realizados na última década são inconclusivos quanto ao efeito da distância nas atividades de desenvolvimento de software. Alguns estudos sugerem que as distâncias física, temporal e cultural têm efeito significativo na coordenação das atividades, levando até mesmo a atrasos no processo de desenvolvimento. Outros estudos sugerem que as diferentes distâncias tem um efeito cada vez menor, pois as empresas estão aprendendo a enfrentar as barreiras da distância. O efeito da distância nas atividades de desenvolvimento de software depende de diversos fatores, como o contexto dos projetos, o histórico das organizações envolvidas, a experiência e a maturidade dos profissionais, entre outros.

EXERCÍCIOS

8.1 Por que o desenvolvimento de software é uma atividade colaborativa?

8.2 Além das duas práticas discutidas no capítulo (processo de desenvolvimento de software e programação em pares), quais outras práticas são inerentemente colaborativas no desenvolvimento de software? Analise os aspectos colaborativos dessas outras práticas.

8.3 Se você fosse contratado para definir um sistema para dar suporte ao desenvolvimento de software que apoie a colaboração dos integrantes de uma equipe de projeto durante o desenvolvimento de software, quais características você listaria para definir tal sistema?

8.4 Além da distância física, quais desafios as equipes distribuídas de desenvolvimento de software enfrentam para colaborar? Justifique sua resposta.

LEITURAS RECOMENDADAS

- Collaborative Software Engineering (Mistrik et al., 2010). Cada capítulo deste livro aborda um tema da área de Engenharia de Software colaborativa incluindo sistemas, métodos ágeis, processo de software, entre outros.
- Guest Editors' Introduction: Cooperative and Human Aspects of Software Engineering (de Souza et al., 2009). Este artigo apresenta uma breve introdução à área de Aspectos Humanos e Colaborativos da Engenharia de Software. O artigo precede uma edição especial da revista IEEE Software sobre este mesmo tema.
- Desenvolvimento Distribuído de Software: Desenvolvimento de Software com Equipes Distribuídas (Audy e Prikladnicki, 2007). Este livro apresenta conceitos da área de Desenvolvimento Distribuído de Software, sendo hoje o único livro em português sobre o tema.
- Distance matters (Olson e Olson, 2000). Este artigo apresenta uma visão geral da pesquisa nos 10 anos precursores à publicação do artigo sobre atividades colaborativas envolvendo

participantes colocalizados e geograficamente distribuídos. O artigo discute o efeito da distância em atividades colaborativas.

REFERÊNCIAS

ALLEN, T. J. Managing the Flow of Technology. Crambridge: MIT Press, 1977, 256 p.

AUDY, J. L. N.; PRIKLADNICKI, R. Desenvolvimento Distribuído de Software. Rio de Janeiro: Elsevier, 2007, 211 p.

CURTIS, B.; KRASNER, H.; ISCOE, N. A field study of the software design process for large systems. Communications of the ACM, v. 31, n. 11, p. 1268-1287, 1988.

DE SOUZA, C. R. B.; SHARP, H.; SINGER, J.; CHENG, L.; VENOLIA, G. IEEE Software, v. 26, n. 6, p. 17-19, 2009.

GONÇALVES, M. K.; DE SOUZA, C. R. B.; GONZALEZ, V. M. Initial Findings from an Observational Study of Software Engineers. In: INTERNATIONAL CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK IN DESIGN, 2009, Santiago. Anais… Santiago: IEEE Press, 2009. p. 498-503.

GONZALES, V. M.; MARK, G. Constant, Constant, Multi-tasking Craziness. In: CONFERENCE ON HUMAN FACTORS IN COMPUTER SYSTEMS, 2004, Vienna. Anais…Austria: ACM Press, 2004. p. 113-120.

MISTRÍK, I; GRUNDY, J.; VAN DER HOEK, A.; WHITEHEAD, J. Collaborative Software Engineering. London: Springer, 2010, 480 p.

OLSON, G.; OLSON, J. Distance Matters. Human-Computer Interaction, v. 15, n. 2, p. 139-178, 2000.

PERRY, D.; STAUDENMAYER, N.; VOTTA, L. People, Organizations, and Process Improvement. IEEE Software, v. 11, n. 4, p. 36-45, 1994.

PRESSMAN, R. Engenharia de Software. São Paulo: McGraw-Hill, 6a ed., 2006. 720 p.