Gerência de Projetos

Por que o Google não usa “branch” de código-fonte?

Quarta-feira, 28/09/2011 às 10h00, por Fabrício Vargas Matos

É muito bom vermos, a cada ano, as nossas empresas crescendo. É uma nova sala aqui, uma leva de novos colaboradores ali, novos projetos e, consequentemente, novos desafios. Procuramos manter nossos procedimentos internos e fazer do nosso ambiente o mais simples possível. Contudo, à medida que crescemos, os desafios gerenciais crescem junto. Poderia falar de muitas questões, mas vou me ater a uma em particular que me chamou a atenção recentemente, ao conhecer um pouco mais sobre o processo de desenvolvimento da Google.

Uma das atividades essenciais em qualquer empresa de software, das menores às maiores, é aquela relacionada à Gerência de Configuração de Software. Em geral, estamos falando do controle de versão do código-fonte, o controle de mudanças e a auditoria das configurações. Um dos nossos produtos mais antigos, o Q-Acadêmico, com mais de 200 mil usuários, teve em 2011 (até o presente mês de setembro) uma média de 6,75 releases por mês. Quase dois por semana. Considerando que o trabalho de preparar um release pode ser grande, especialmente se os procedimentos adotados pelos desenvolvedores não forem bem pensados, suponho que esse seja um assunto relevante para qualquer gerência de desenvolvimento. Só para lembrar, nós não gerenciamos apenas componentes de software e suas dependências, o que por si só já pode se tornar bem complexo se a quantidade for grande. Mas nós gerenciamos também dados de configuração e meta-dados, que precisam ser atualizados sem prejuízo para os dados dos clientes. À medida que a quantidade de sistemas e de desenvolvedores que alteram esses sistemas aumenta, a gerência de configuração e os processos de engenharia se tornam mais desafiadores, podendo gerar grandes gargalos, se não forem bem pensados.

Por exemplo, é comum estarmos desenvolvendo alguma nova funcionalidade não trivial - algo que levará algumas semanas para ser concluído, e antes disso precisarmos liberar um release com pequenos ajustes que não podem esperar. Por se tratar de uma situação bastante comum, precisamos ter um procedimento de trabalho que facilite a preparação do release. Geralmente, o procedimento que adotamos é criar um *branch* para as implementações maiores (mais de 1 semana, por exemplo), enquanto os ajustes mais pontuais são feitos diretamente no *trunk*. Dessa forma, ele pode ser rapidamente liberado para os clientes. Contudo, quando as alterações maiores, que estão no *branch,* forem concluídas, um trabalho de *merge* terá de ser realizado, e é aí que mora o perigo. Dependendo das mudanças, alguns conflitos podem aparecer na hora do *merge*. Desde alterações conflitantes em uma mesma classe, ou método - que por ser bem localizado tende a ser mais simples de corrigir, como problemas mais sérios, relacionados a dependências, que deixam de funcionar após o *merge*. E isso impede a própria compilação, exigindo alguém com uma boa visão do todo para avaliar como deve ser resolvido o problema, tendo, inclusive, que ajustar métodos de teste, mas tomando cuidado para não afetar nenhum comportamento. Esse não é um trabalho para qualquer um, pois aumenta o custo operacional e introduz riscos adicionais ao processo por causa das possibilidades de falha humana.

Outra questão comum é a gestão do controle de versões de bibliotecas compartilhadas entre projetos. Devemos fazer referência aos módulos compilados, ou ao código-fonte? Qual é a

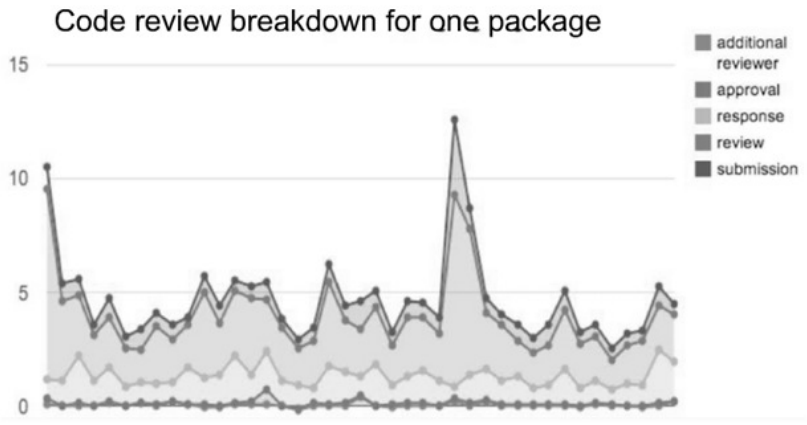
melhor forma de compartilhar o código-fonte sem replicá-lo, mas permitindo que cada projeto possa utilizar diferentes versões da mesma biblioteca e incluí-las nos branches e nas tags? Qualquer caminho tem seus prós e contras, e nem sempre é simples fazer uma escolha.

Enfim, se para nós, que temos menos de 50 projetos ativos (sistemas), isso já é um problema considerável, fico imaginando uma corporação como a Google, com seus 2 mil projetos e 5 mil desenvolvedores, alterando mais de 20 arquivos por minuto, como nos relata Ashish Kumar, *Engineering Manager* do Google, em sua palestra Development at the Speed and Scale of Google. Sempre fico penando sobre como lidaríamos com esses problemas se fôssemos vinte vezes maior do que somos hoje. Alguns defendem que a solução é continuar pequeno. Não é nosso caso. Nós não vamos deixar de crescer para resolver um problema de engenharia de software. Encaro todo problema que enfrentamos como um desafio a ser resolvido, e nosso trabalho é propor soluções inteligentes pra eles.

Analisando a estratégia adotada pela Google em seu processo de desenvolvimento de software (assista à palestra do Sr. Kumar para mais detalhes), alguns pontos me chamaram a atenção, principalmente o pragmatismo com o qual eles fugiram do senso comum e deram soluções, às vezes até radicais, para manter tudo mais simples. Seguem alguns pontos-chave que observei na palestra:

1. Possuem extensas listas de testes automatizados sendo executados a cada alteração. Sistemas internos apontam todos os testes que são afetados pela mudança, para que o desenvolvedor garanta que tudo continue funcionando antes de subir suas alterações. A integração contínua executa 120 milhões de casos de teste por dia, e esse número continua crescendo.
2. Todas as dependências são compiladas na hora do build, sempre a partir do *trunk*. Com isso, qualquer alteração que tenha impacto em outros módulos será reportada na hora. *Feeback* mais rápido significa menos trabalho depois e adiar a detecção do problema pode aumentar, significativamente, o trabalho à medida que diferentes conflitos se misturam.
3. Melhoria contínua dos processos de engenharia de software:
    1. “Não podemos melhorar o que não medimos”
    2. Cultura focada em gerar dados e métricas que permitam melhoria contínua
    3. Ter como meta tornar as ferramentas invisíveis ao trabalho cotidiano (automação)
    4. Revisão de código obrigatória. Através de uma ferramenta centralizada, revisores são automaticamente selecionados sempre que alguma alteração é feita. Também existem “donos de sub-árvores” do código, responsáveis por manter a consistência e a correção do código sob sua responsabilidade.

O gráfico abaixo mostra um histórico de alterações de um determinado pacote, apontando o tempo gasto com cada etapa do processo de revisão de código, desde a alteração (submission) até a aprovação (approval).



Note que o tempo de review é bem longo, mas medi-lo permite que ações sejam tomadas para otimizar o trabalho de revisão, como automatizar parte da análise de código com Lint (desenvolvedor C#. pense no Resharper, por exemplo, apontando código fora do padrão estabelecido).

A Google também investe muito em infraestrutura para os desenvolvedores. O trabalho

pesado de compilação, por exemplo, é executado em clusters, e não na estação de trabalho do desenvolvedor. Eles disponibilizam mais de 10 mil núcleos, totalizando 50TB de RAM. Paralelizam e distribuem sempre que possível. Eles podem!

Essa atitude da Google tem me levado a repensar algumas coisas. Às vezes, quanto mais complicado um problema, mais simples e direta deve ser a solução. A estratégia de sempre trabalhar com dependência, via código-fonte e sempre utilizar o *trunk* é um caminho radicalmente diferente do que estou acostumado (utilizar módulos compilados para não dar checkout em tudo e trabalhar com branches + merge), mas para tomar esse caminho precisaremos repensar outros aspectos do nosso processo de software.

Um ponto bastante enfatizado na palestra de Ashish Kumar é a estratégia deles de *code review*, que é colocada claramente como um fator chave de sucesso, e que, para ele, poderia ser facilmente utilizada por uma empresa pequena.

Há muitos pontos obscuros acerca desse assunto, mas prefiro parar por aqui. Se puder e quiser, compartilhe conosco sua experiência com a gerência de configuração.

Fabrício Vargas Matos é Bacharel em Ciência da Computação e Mestre em Informática pela Universidade Federal do Espírito Santo (Ufes). Atualmente é Diretor Executivo da Qualidata, e escreve artigos para o blog da empresa.

Página do autor **Email** **Website**

Leia os últimos artigos publicados por fabricio_vargas_matos

Cuidado: mocks podem comprometer a legibilidade dos testes

Mantenha a mente aberta. TDD nem sempre é o melhor!

Specification Pattern em .NET: vale a pena?

Devemos "pensar lá na frente" no início de um projeto de software?