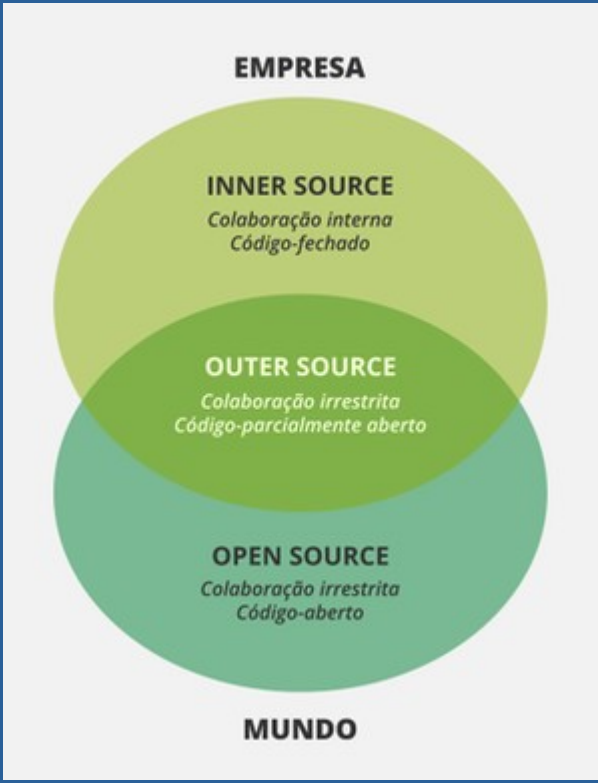
COMEÇANDO COM INNERSOURCE

CHAVES PARA COLABORAÇÃO E PRODUTIVIDADE
DENTRO DE SUA EMPRESA



ANDY ORAM

Traduzido por Flávio Gomes da Silva Lisboa, M.Sc.

Traduzido do original de 2015 da O’Reilly Media

O texto original está disponível em:
https://learning.oreilly.com/library/view/getting-started-with/9781098107345/

Tradução autorizada pela O’Reilly Media sob as condições de que este material seja fornecido online sem qualquer cobrança.

Fonte da imagem da capa:
http://intra.serpro.gov.br/tema/artigos-opinioes/como-innersource-pode-envolver-mais-o-usuario-na-construcao-do-software

Índice

Uma abordagem robusta para a colaboração em equipe

Inspiradas pela disseminação do software de código aberto nas áreas de sistemas operacionais, computação em nuvem, estruturas JavaScript e outras, várias empresas estão imitando as práticas do poderoso movimento de código aberto para criar uma colaboração corporativa interna sob a rubrica InnerSource. Nessas páginas, você lerá sobre a experiência do principal facilitador de comércio na Internet PayPal e verá como a InnerSource pode beneficiar engenheiros, gerenciamento e departamentos de marketing / RP[1].

Para entender o apelo do gerenciamento de projetos da InnerSource, considere o que tornou o desenvolvimento de software de código aberto tão bem-sucedido:

- Os programadores compartilham seu trabalho com um público amplo, em vez de apenas com um gerente ou equipe. Na maioria dos projetos de código aberto, qualquer pessoa no mundo é livre para visualizar o código, comentá-lo, aprender novas habilidades examinando-o e enviar alterações que acham que irão melhorá-lo ou personalizá-lo de acordo com suas necessidades.
- Novos repositórios de código (branches) baseados no projeto podem ser feitos livremente, para que sites com usos imprevistos para o código possam adaptá-lo. Geralmente, existem regras e suporte técnico para recolocar ramos diferentes no ramo mestre original.

1 Relações Públicas

- Pessoas em grandes distâncias geográficas, em momentos separados, podem trabalhar no mesmo código ou contribuir com arquivos de código diferentes para o mesmo projeto.
- A comunicação tende a ser escrita e postada em sites públicos, em vez de compartilhada informalmente de boca em boca, o que fornece um histórico do projeto, bem como oportunidades de aprendizagem para novos membros do projeto.
- Escrever testes de unidade torna-se uma tarefa-chave de programação. Um “teste de unidade” é um pequeno teste que verifica um comportamento particular e isolado, como rejeitar uma entrada incorreta ou tomar o desvio apropriado sob certas condições. No código-fonte aberto e no InnerSource, os testes são feitos constantemente conforme as alterações são verificadas, para proteger contra falhas durante as execuções de produção.

InnerSource difere do código aberto clássico por permanecer dentro da visão e controle de uma única organização. A “abertura” do projeto se estende a muitas equipes dentro da organização. Isso permite que a organização incorpore segredos comerciais diferenciadores no código sem medo de que sejam revelados a estranhos, enquanto se beneficia da criatividade e das diversas perspectivas oferecidas por pessoas em toda a organização.

Frequentemente, a organização opta por compartilhar partes de um projeto InnerSource com o público, efetivamente transformando-os em código aberto. Quando as tecnologias e práticas de gerenciamento de código aberto são usadas

internamente, mover o projeto para uma arena pública se torna muito mais fácil.

As vantagens de adotar uma estratégia InnerSource incluem:

- A reutilização de código em toda a organização cresce imensamente. Os programadores de cada equipe podem compreender o código e a arquitetura dos módulos desenvolvidos por outras equipes e contribuir com o código.
- Essa colaboração entre equipes torna-se relativamente fácil.
- O código fornecido raramente precisa ser reescrito pela equipe que o recebe e nenhuma discussão é necessária em um nível de gerenciamento superior.
- O desenvolvimento se torna mais rápido à medida que os programadores aprendem a usar testes de unidade, cobertura de código e integração contínua para remover bugs em um estágio inicial de desenvolvimento. Os comentários escritos trocados entre os membros da equipe, embora ocupem algum tempo, mais do que se pagam, ajudando novos programadores a aprender o sistema mais rápido.
- Os programadores aprendem a documentar melhor seu código, tanto formalmente (como comentários e documentação no código) quanto informalmente (em listas de discussão). A documentação fornece um histórico do projeto e ajuda pessoas de fora a entendê-lo para que mais pessoas possam contribuir com ele.

Onde os princípios de código aberto funcionam

Antes de pensar sobre o que a InnerSource pode realizar para sua organização, é importante considerar onde a fonte dessas práticas – o movimento do software livre e de código aberto – funcionou bem e onde historicamente tem sido menos bem-sucedido.

Colaboração interorganizacional

O desenvolvimento de software de código aberto atravessa as fronteiras de equipes, empresas e nações, permitindo que indivíduos em todo o mundo colaborem em avanços de computação importantes como Linux e Apache. O estereótipo de hackers que criam código em um porão em algum lugar está desatualizado (se é que alguma vez foi relevante): os maiores contribuintes para projetos centrais de código aberto como o Linux agora trabalham para grandes corporações.

Muitos projetos de código aberto importantes começam como fruto da imaginação de programadores independentes, mas são frequentemente adotados por grandes empresas. Em outros casos, os projetos de código aberto surgem de dentro dessas empresas, incluindo o PayPal. O que continua sendo crucial para seu sucesso de código aberto é que as pessoas dentro de qualquer organização permitem que estranhos desempenhem um papel importante no desenvolvimento, teste,

documentação e promoção do software, criando o mais nivelado possível para todos os colaboradores.

A diferença entre desenvolvimento geograficamente disperso e programação ágil

A atividade transfronteiriça do código aberto contrasta fortemente com as abordagens populares de desenvolvimento corporativo, como Agile e Scrum. Os últimos métodos de desenvolvimento foram projetados explicitamente para equipes que trabalham em um único escritório, comunicando-se face a face. Os desenvolvedores de tais projetos não devem apenas trabalhar juntos, mas também devem se reunir com os usuários do software e outras partes interessadas regularmente. Todo esse compartilhamento informal de informações reduz a necessidade de documentação e requisitos formais. Mas tais métodos de desenvolvimento não funcionam tão bem em ambientes maiores do que um único trabalho
local.

Se o Agile e o Scrum pagam um preço em escalabilidade, o desenvolvimento de código aberto paga um preço em velocidade. Projetos de código aberto simplesmente não podem ser movidos tão rapidamente quanto equipes dedicadas trabalhando em um único escritório. Esse preço obviamente faz uma grande diferença para empreendimentos iniciais com prazos curtos (às vezes prazos extremamente curtos). A diferença também tem um efeito sutil sobre os tipos de software que são adequados para cada modelo de

desenvolvimento, como você verá em “Valor para a infraestrutura de software”.

Apesar das limitações na velocidade de progresso, muitos projetos de código aberto mantêm prazos rígidos e lançamentos previsíveis. No entanto, eles geralmente sacrificam recursos para previsibilidade. Alguns aprimoramentos que estão planejados para um determinado lançamento podem ter que ser adiados para evitar atrasar o prazo.

A programação ágil não precisa ser inimiga do desenvolvimento de código aberto – na verdade, a literatura relata equipes que combinam os dois[2]. Mas entender seus diferentes requisitos e suposições pode dar a você uma melhor compreensão do código aberto.

Teste e Desenvolvimento Contínuo

Em geral, os projetos de código aberto mantêm a qualidade e a confiança entre os colaboradores, estabelecendo um sistema rigoroso de teste objetivo de cada contribuição. Muitos projetos de código aberto adotaram práticas comumente recomendadas, como teste de unidade e integração contínua, e desenvolveram modelos organizacionais formais para garantir seu uso. Os desenvolvedores de código aberto transformaram o controle de qualidade em uma ciência.

Embora o desenvolvimento orientado a testes não seja necessário para o código aberto, o teste de unidade é levado muito a sério. Independentemente de um projeto de código

2 Stol, KJ, P. Avgeriou et al. “Key Factors for Adopting Inner Source.” ACM Transactions on Software Engineering Methodology (TOSEM) (2014): Vol 23, No 2.

aberto manter ou não uma equipe de garantia de qualidade, espera-se que cada programador escreva testes de unidade para garantir a qualidade de seu próprio código. Ferramentas e protocolos para confirmar mudanças requerem que os testes sejam executados e produzam um resultado limpo antes que o código possa ser aceito no repositório. O processo oferece garantia não apenas de que o código funciona da maneira como foi prometido, mas também de que não reage mal com alguma outra parte do código em outro lugar do projeto.

Técnicas modernas, como ferramentas de cobertura de código e análise estática, são usadas com menos frequência em comunidades de código aberto e têm sido adotadas com dedicação no PayPal.

No geral, o teste e a integração contínua desempenham duas funções importantes. Acima de tudo, é claro, eles evitam que o produto se quebre. Mas também ajudam a identificar bons programadores que podem receber maior responsabilidade pelo projeto. Por exemplo, obter um grande número de commits aceitos no repositório é geralmente um pré-requisito para obter o cobiçado status de um committer confiável. Os committers confiáveis revisam e aprovam o trabalho de outros programadores e podem fazer alterações sem a necessidade de tal aprovação. Eles também fornecem orientação para contribuidores promissores cujo código ainda não atende aos padrões de qualidade.

A Importância da Documentação

Outro princípio do desenvolvimento de código aberto, decorrente da dispersão de seus praticantes ao longo da geografia e do tempo, é a documentação completa. Projetos de

código aberto tendem a ser fracos na documentação do usuário (uma falha que eles compartilham com a maioria dos projetos proprietários), mas os desenvolvedores escrevem obsessivamente uns para os outros sobre suas suposições, decisões e implementações.

Essa prática representa outro contraste com a programação Agile, que exige alguma documentação, mas geralmente favorece o "software funcional em vez da documentação" (uma cláusula frequentemente citada do Manifesto Agile). Deve-se observar que o código-fonte funcional também é o padrão platina na comunidade de código aberto (assim como nos órgãos de padrões dominados por desenvolvedores), mas essas comunidades ainda consideram importante documentar o que foi feito.

Cada faceta da comunicação em software de código aberto é escrita. Os comentários no GitHub são os principais impulsionadores de muitos projetos. Projetos de código aberto dependem de listas de e-mails para discussões que levam a todas as decisões, e você costuma ouvir: "Se não está na lista de e-mails, não aconteceu".

Um valor de documentar decisões e detalhes de implementação é que um histórico do projeto é criado para qualquer novo que queira ingressar. Qualquer pessoa que dedicar um tempo para examinar os arquivos de discussão pode aprender a cultura do projeto e as práticas recomendadas.

Cada colaboração pressupõe que os participantes compartilham um idioma comum. Dada a diversidade geográfica dos projetos de código aberto, é claro, o inglês se tornou a língua franca praticada por todos os programadores (embora haja projetos de código aberto significativos em outras línguas também). Tomar decisões por meio da comunicação escrita desempenha um papel democratizador aqui, porque

mais pessoas podem aprender a ler e escrever uma língua estrangeira do que a falar fluentemente. Essas pessoas poderiam participar mais em projetos de código aberto do que em equipes bem unidas usando uma linguagem que não conhecem bem.

Valor para infraestrutura de software

O software de código aberto tradicionalmente funcionou bem para as camadas inferiores da pilha de software: sistemas operacionais e hipervisores, ferramentas usadas por programadores como compiladores e editores, software de segurança (que se beneficia de análises de código aberto para detectar fraquezas) e outras coisas escondidas onde o usuário final não o vê.

Em contraste, a interface do usuário (IU) tem se mostrado obstinadamente resistente ao desenvolvimento de código aberto. É difícil apontar para uma ferramenta de usuário final de código aberto que obteve adoção em massa. O Mozilla Firefox é um exemplo raro.

Olhando para trás, para o contraste entre o código aberto e o desenvolvimento ágil (“A diferença entre o desenvolvimento geograficamente disperso e a programação ágil”), você encontra uma pista do motivo dessa falta de sucesso. O modelo Agile foi amplamente adotado porque mantém os programadores em contato próximo com os usuários. Os desenvolvedores recebem feedback do usuário e podem começar a trabalhar com ele em questão de dias. A maioria dos projetos de código aberto envolve usuários finais de maneiras relativamente antiquadas, como por meio de versões alfa e beta.

A ênfase no teste de unidade (“Teste e Desenvolvimento Contínuo”) também marca o código aberto como apropriado para infraestrutura. Cerca de cinco anos atrás, o especialista em Agile Mike Cohn descreveu uma pirâmide de teste que coloca várias camadas de infraestrutura sob uma pequena camada de interface do usuário. Ele exigiu testes de unidade para o maior número possível de camadas do software, reduzindo a dependência de testes funcionais.

A camada de IU, por outro lado, é muito difícil de verificar por meio de testes de unidade, e esses testes mostram confiabilidade e valor decrescentes nessa camada. É aqui que o teste funcional entra para verificar a operação geral de um sistema e garantir que cada ação realizada pelo usuário final produza o resultado desejado.

Portanto, o processo de teste que sustenta a qualidade e a confiança no código aberto é fraco no nível da interface do usuário, o que pode ser outro motivo pelo qual as ferramentas de código aberto são menos populares nesse nível.

Encontrando o nível certo para código aberto

O software livre e de código aberto tem sido tradicionalmente contrastado com o software proprietário (especialmente Microsoft Windows e banco de dados Oracle). Uma abordagem mais comum para o desenvolvimento proprietário hoje em dia é software como serviço (SaaS). Com este modelo de entrega de software, você pode usar as diretrizes nesta seção para determinar o que deve ser desenvolvido em uma maneira InnerSource ou open source, e o

que fazer de uma maneira mais fechada usando Agile ou alguma outra metodologia orientada para a equipe.

Muitas empresas misturam esses modelos. Essas empresas podem ser chamadas de "núcleo fechado", porque mantêm o software fundamental para seus negócios por trás do SaaS, mas compartilham software de infraestrutura gratuitamente. Esse acordo beneficia a comunidade de programação mais ampla, torna mais fácil recrutar e treinar funcionários e traz os benefícios de contribuições externas.

Como o PayPal adotou a InnerSource

O caminho do PayPal para a InnerSource envolveu uma série de decisões corporativas históricas e em grande escala. Ela adotou o modelo como uma das várias mudanças feitas conscientemente nas ferramentas e na cultura corporativa, como costuma ser o caso.

No caso do PayPal, as mudanças que precederam a adoção do InnerSource incluíram:

- Uma busca por tecnologias que promovam um desenvolvimento mais rápido (substituindo Java por JavaScript e Node.js em muitos projetos, por exemplo)
- Um consequente interesse em uma melhor compreensão das comunidades de código aberto e modelos de desenvolvimento vinculados às tecnologias recentemente adotadas
- Uso do GitHub para colaboração interna e externamente
- Maior preocupação com a qualidade

O PayPal, portanto, adotou algumas tecnologias de código aberto e até mesmo abriu o código de seu próprio código antes de experimentar o modelo InnerSource. Outras empresas podem tomar o caminho inverso: elas podem experimentar as ferramentas e práticas do código aberto dentro de suas próprias paredes antes de produzir qualquer código-fonte aberto, embora esse caminho seja geralmente considerado mais desafiador. Em ambos os casos, a familiaridade com as ferramentas de código aberto, junto com sites como o GitHub, que facilitam a colaboração de uma maneira de código aberto, é crucial.

Embora as várias mudanças no PayPal ocorreram juntas e claramente tiveram impactos umas sobre as outras, as próximas seções descreverão cada mudança individualmente, com foco nas atividades específicas de cada um.

Começando no Limite

A InnerSource no PayPal começou com engenheiros de vendas regionais, trabalhando fora dos Estados Unidos, que modificaram o código voltado para o usuário para oferecer suporte às preferências de uso local e, às vezes, para oferecer suporte a promoções regionais. Esses programadores não conseguiram que suas alterações fossem aceitas pelas equipes principais no prazo necessário. Frequentemente, essas mudanças exigiam a intervenção de um VP em algum lugar para exigir que fossem fundidas, após o que um engenheiro central em apuros reescreveria as alterações enviadas antes de fundi-las.

Ao mesmo tempo, a documentação inadequada estava sendo preenchida por meio de trocas de e-mail realizadas com

urgência entre desenvolvedores de equipes diferentes. Embora essas mensagens contivessem informações valiosas, elas não foram vistas pela maioria dos desenvolvedores. Muitas organizações regionais iniciaram esforços independentes para recortar e colar essas mensagens em wikis.

Vários engenheiros regionais foram levados de avião para a sede de San Jose para passar um mês aprendendo diretamente com as equipes principais, mas esse método de inicializar committers confiáveis não espalhou o conhecimento amplamente o suficiente e não valeu o investimento. Nenhuma documentação escrita desenvolvida a partir dele. Depois que os engenheiros voltaram para seus países de origem, foram forçados pela distância a buscar orientação central por meio de consultas por escrito.

Eventualmente, foi sugerido que o InnerSource poderia ser a melhor maneira de fazer isso. As equipes descobriram que a InnerSource realmente simplificou seus projetos, e os committers confiáveis no lado central acharam o processo muito melhor. Os engenheiros remotos podem obter amostras de código do GitHub.

Além de aprender como trabalhar melhor com colaboradores regionais, as equipes principais começaram a obter pistas (por meio de sua orientação de colaboradores regionais) sobre onde refatorar o código principal para aumentar a modularidade e a compreensibilidade. Eles também foram capazes de perceber padrões entre o trabalho realizado em diferentes regiões, o que lhes permitiu promover engenheiros regionais experientes como mentores de seus colegas em outras regiões.

Um processo de desenvolvimento mais rápido

O PayPal começou com uma plataforma C ++ relativamente monolítica. Embora alguns elementos legados ainda estejam em C ++, o novo desenvolvimento por um tempo foi feito de uma forma mais modular usando Java Spring. Mesmo após essa mudança, como disse o desenvolvedor Jeff Harrell, muito "conhecimento tribal" foi incorporado a cada produto. Demorava várias semanas para implantar até mesmo uma pequena mudança, e cada nova contratação exigia um período de treinamento de seis semanas.

Três anos atrás, o PayPal fez uma grande mudança ao adotar o Node.js. De acordo com Poornima Venkatakrishnan, um desenvolvedor Node.js no PayPal, a plataforma foi usada lá primeiro para prototipagem. Felizes com os resultados, os desenvolvedores queriam implantá-lo na produção. Primeiro, eles tentaram um experimento em que as equipes desenvolveram a mesma funcionalidade lado a lado na plataforma Java Spring e no Node.js. A empresa comparou as plataformas com base no tempo de desenvolvimento, número de linhas de código e número de engenheiros necessários para o desenvolvimento. Node.js foi um vencedor óbvio.

O anúncio de que o PayPal estava adotando o Node.js preocupou muitos programadores, especialmente aqueles que se lembravam da mudança de C ++ para Java como longa e cansativa. No entanto, o PayPal demonstrou que a mudança para Node.js foi completamente diferente. Ele agendou apenas dois dias para o treinamento de transição. De acordo com Harrell, os participantes perceberam rapidamente que estavam entrando em um mundo novo, vibrante e empolgante. O PayPal

também teve a sorte de ter na equipe um líder na área de JavaScript, Douglas Crockford, para fazer o treinamento. (Crockford é autor de um livro muito popular, *JavaScript: The Good Parts*, e criador de um vídeo O'Reilly *JavaScript Master Class*.)

Muitos novos recursos de desenvolvimento do PayPal ainda usam Java, mas essas equipes adotaram as práticas do InnerSource que foram iniciadas pelas equipes do Node.js.

Envolvendo-se com código aberto

O treinamento para a mudança para Node.js foi rápido e indolor, porque muito material de instrução estava disponível online e porque a adoção do software exigia menos customização do que a mudança anterior para Java. Basicamente, dois dias de treinamento foram suficientes para preparar a equipe para continuar seus próprios processos de aprendizagem.

De acordo com Harrell, demorou algum tempo para transmitir aos programadores do PayPal que eles poderiam simplesmente pesquisar informações básicas em sites como o StackOverflow (ou apenas usar um mecanismo de busca) em vez de perguntar aos programadores seniores em uma lista de mala direta interna do PayPal. Mas, eventualmente, as consultas por informações se ampliaram para um envolvimento de todo o coração com as comunidades de código aberto.

A maioria das empresas acha que a adoção de tecnologias de código aberto – que originalmente podem ser atraentes simplesmente porque são de alta qualidade e gratuitas – leva a uma entrada sustentada nas comunidades que produzem as tecnologias.

Isso aconteceu com o PayPal quando eles adotaram o Node.js. Agora é uma tecnologia bastante madura. Alguns comentaristas chegam a sugerir que o Node.js ultrapassou seu pico e que os programadores de JavaScript estão procurando alternativas. (O ritmo de mudança nas tecnologias de código aberto, como você verá mais adiante nesta seção, altera a tomada de decisão sobre quais adotar.) Mas o PayPal chegou em um estágio inicial, onde havia muito espaço para desenvolver ferramentas úteis para a comunidade Node.js. O PayPal também se tornou um membro muito ativo da fundação Node.js e do Comitê Técnico ECMAScript 39.

Para promover o duplo objetivo de autodidatismo do programador e produtividade imediata entre os novos contratados, o PayPal trouxe com entusiasmo outras ferramentas populares de código aberto durante a mudança para JavaScript e Node.js. Por exemplo, Selenium é amplamente usado para teste, Jenkins para integração contínua e ferramentas como TestNG, JUnit e Mockito para teste de Java. Ao longo do caminho, o PayPal desenvolveu e abriu o código-fonte de uma série de ferramentas compatíveis com os componentes de código-fonte aberto, como Nemo (originalmente escrito por Matt Edelman), SeLion e Illuminator.

Como o PayPal adotou o Node.js quando estava no início do desenvolvimento e carecia de certas ferramentas, a empresa desenvolveu o Kraken, uma estrutura para controlar as arquiteturas Node.js. Harrell diz que a equipe decidiu desenvolver a estrutura como código-fonte aberto para garantir que fosse de valor geral, livre do conhecimento "tribal" que Harrell tinha visto em outro código do PayPal. O Kraken se tornou provavelmente o projeto de código aberto de maior sucesso.

Venkatakrishnan lista as seguintes práticas de código aberto que entraram no PayPal neste momento:

- Definição de padrões de alta qualidade. Antes de o código ser confirmado, os testes de cobertura de código devem ser executados em pelo menos 90% dele. Quando alguém envia um commit, ele aciona uma reconstrução automática para garantir que seja de qualidade suficiente para ser mesclado.
- Exigir documentação para todos os códigos nos próprios repositórios.
- Conduzir todas as discussões no GitHub para permitir a colaboração e facilitar a entrada de pessoas fora da equipe. Há um repositório GitHub privado para projetos internos, bem como repositórios GitHub públicos para código-fonte aberto.
- Fazer com que todos sintam que podem inovar.
- Ensinar aos engenheiros que nem todo conhecimento precisa vir de uma equipe. Com o software de código aberto, a ajuda pode vir de qualquer lugar do mundo. Além disso, contribuir com projetos públicos de código aberto adiciona mais visibilidade à empresa e mostra seu compromisso com a comunidade.
- Cultivar orgulho pelo trabalho realizado em equipes, incentivando-os a falar sobre isso em conferências e escrever sobre isso no blog de engenharia do PayPal.

Harrell destaca que quanto mais alto se chega em uma pilha de software, mais voláteis são as tecnologias. A mudança no desenvolvimento de software está em andamento, especialmente com tecnologias de código aberto. Portanto, o PayPal incentiva uma equipe a avaliar constantemente suas

tecnologias. O aprendizado contínuo é esperado de cada desenvolvedor.

Inevitavelmente, a avaliação de ferramentas alternativas leva a uma diversidade de ferramentas de uma equipe para outra. O desenvolvedor Matt Edelman aponta que esse fenômeno é particularmente comum em código aberto, porque os desenvolvedores gostam de desenvolver ferramentas nas linhas sugeridas pelo patriarca do Unix Ken Thompson: muitos pequenos programas trabalhando juntos.

Assim, as equipes usam uma variedade de tecnologias de ponta, como React e Angular. Espera-se que os programadores façam um projeto de brinquedo para experimentar uma ferramenta antes de recomendá-la e para comparar alternativas cuidadosamente antes de fazer uma escolha.

As arquiteturas de código aberto geralmente apresentam plug-ins para permitir implementações alternativas de tecnologias que estendem sua funcionalidade. Edelman diz que o PayPal desenvolve alguns plug-ins Kraken internamente (por meio do modelo InnerSource) e outros como código aberto.

Colaboração GitHub

Uma empresa ocupa um lugar exclusivamente importante no desenvolvimento de código moderno: GitHub. Originalmente uma plataforma SaaS para facilitar a vida dos desenvolvedores que usam o software de controle de versão Git, o GitHub evoluiu para uma plataforma de colaboração sofisticada com ferramentas que os desenvolvedores consideram indispensáveis. Dois recursos sobre este tópico incluem o livro *Git for Teams* e o vídeo *Collaborating with Git*.

Além de colocar código para projetos de código aberto como Kraken no GitHub, o PayPal configurou um repositório GitHub Enterprise interno para que os programadores dentro da empresa possam colaborar exatamente como os usuários públicos do GitHub. As revisões, confirmações e testes de código ocorrem de forma open source, mas no repositório interno. Mais importante ainda, cada equipe se acostuma a aceitar novos códigos de programadores do PayPal fora da equipe.

Antes da mudança para GitHub e InnerSource, de acordo com Edelman, os programadores do PayPal se sentiam competentes para contribuir com pequenas correções de bugs para o trabalho de outra equipe. Se eles contribuíssem com pedaços maiores, a equipe que aceitava o código iria examiná-lo e freqüentemente reescreveria tanto que haveria pouca vantagem sobre fazer o código do zero. Os gerentes de alto nível frequentemente teriam que se envolver para negociar a aceitação do código entre as equipes.

Hoje em dia, novas funcionalidades substanciais podem ser verificadas sem necessidade de reescrita. Um dos motivos é o novo enfoque na documentação. Edelman enfatiza que os projetos da InnerSource devem se apoiar em uma base de programação mais ampla do que alguns programadores privilegiados que entendem profundamente o sistema. Qualquer pessoa pode estudar um projeto e sugerir mudanças importantes.

Também graças à documentação (tanto no código quanto nos comentários do GitHub), os desenvolvedores reconhecem a necessidade de mudanças na arquitetura. Se você tiver que explicar quatro vezes a várias pessoas por que algo é complicado e contra-intuitivo, comece a pensar em mudar isso.

Capacitar programadores em todo o país também promove maior crescimento intelectual e satisfação no trabalho. Os programadores pensam de forma mais abrangente sobre o design do código, enquanto aprendem novas habilidades em fazer revisões de código, testar e escrever documentação. Edelman diz que a InnerSource "aumenta o jogo de todos".

Edelman menciona um bug relacionado à segurança que apareceu no módulo de infraestrutura do núcleo. Uma função não estava seguindo a especificação (RFC) com precisão. A princípio, muitos e-mails foram trocados, com pessoas de fora da equipe de infraestrutura pressionando-os para fazer uma correção. Mas então um estranho – para quem segurança não era uma especialidade – deu uma olhada no RFC e percebeu que poderia cuidar da correção sozinho. Ele apresentou uma correção que funcionou e foi rapidamente incorporada pela equipe de infraestrutura central. Esta anedota ilustra dois princípios: a importância dos programadores tomarem a iniciativa e o valor de seguir padrões bem documentados.

Melhoria da Qualidade

Ao mesmo tempo em que transferia grande parte de sua plataforma para Node.js e adotava práticas de código aberto, o PayPal se comprometia com a melhor qualidade. De acordo com o líder de garantia de qualidade Doug Simmons, as etapas em direção a essa meta no PayPal incluem:

- Mais testes de unidade
- Integração contínua
- Revisões de código
- Mais relatórios de cobertura de código

- Análise estática

Nem todas as conversas precisam ser públicas, mas o processo de comentários no GitHub melhorou a qualidade de várias maneiras. Os committers confiáveis atuam efetivamente como mentores para os programadores mais novos, na esperança de que seu código seja aceito no repositório. Os comentários de vaivém entre os remetentes e os confirmadores confiáveis são educacionais tanto para os remetentes quanto para outros programadores que podem assistir a esses diálogos e aprender com eles.

Mudança de cultura

Um movimento corporativo tão significativo quanto a adoção da InnerSource exige sensibilidade ao lidar com os medos e expectativas dos funcionários. O PayPal contratou especialistas externos em desenvolvimento de código aberto para ajudar na transição.

Observadores de muitas empresas parecem concordar que a mudança cultural mais importante é dar aos funcionários a confiança para contribuir com códigos para outras equipes. Documentação completa e um bom mentoreamento podem conseguir isso - mas, como Edelman aponta, a equipe tem que mudar seus hábitos de reclamar de bugs para entrar e fazer correções.

Harrell se lembra de ter persuadido os membros da equipe a permitir que outros programadores do PayPal fora da equipe fizessem contribuições. Os membros da equipe geralmente objetavam que acabariam gastando todo o tempo examinando o código externo e não escrevendo o seu próprio. Em um estudo de outra empresa, os programadores

preocuparam-se com o fardo que novas contribuições representariam para a manutenção futura[34].

Como você já sabe, essas desvantagens se mostraram menos assustadoras do que o esperado, porque o teste rigoroso e o processo de construção garantiram a qualidade das contribuições. Em qualquer caso, Harrell disse a eles, a verificação das contribuições exigia habilidades sofisticadas próprias. E as contribuições multiplicaram o valor do código criado pela equipe.

Edelman escreve: “Espera-se que os autores do módulo façam a curadoria de seu software, estando abertos a comentários e mudanças da comunidade, ao mesmo tempo que reforçam os padrões de qualidade e consistência”. Ele diz que os desenvolvedores evoluíram de reclamar sobre a tarefa de escrever testes ou fazer revisões de código para insistir neles.

Na área de qualidade, quando os programadores enviam relatórios de bugs do InnerSource, eles vão inicialmente para a pessoa responsável pelo módulo. Esse programador pode então repassá-los aos contribuidores originais do código afetado.

Arquiteturas modulares e APIs bem definidas também foram identificadas por muitas equipes de programação como cruciais para encorajar contribuições de fora da equipe.

Como as práticas de desenvolvimento da InnerSource são essencialmente as práticas de código aberto mais populares, abrir o código de um projeto que foi desenvolvido dentro da empresa é bastante fácil. Tecnicamente, tudo que a equipe precisa fazer é mover o código para um repositório público e começar a lidar com relatórios de bugs externos e

3 Ibid.

4 Stol, KJ, AB Muhammad, et al. “A comparative study of challenges in integrating Open Source Software and Inner Source Software.” Information and Software Technology 53 (2011): 1319–1336.

contribuições de código da mesma forma que tem lidado com essa entrada de outros membros de sua empresa.

No entanto, algumas barreiras legais podem atrapalhar. Se os programadores incorporaram código externo ao projeto, o departamento jurídico deve examinar todas as licenças para garantir que eles tenham o direito de abrir o código e que sua licença seja compatível com as licenças do código que incorporaram. Alguma revisão de marca também pode estar envolvida. No PayPal, a equipe jurídica consultou especialistas em código aberto e desenvolveu um processo baseado na web para realizar as etapas necessárias.

Os programadores modernos aprendem a prosperar na mudança. Além de novas tecnologias e ferramentas, uma mudança de cultura pode ajudá-los a se manterem ágeis e a manter as habilidades atualizadas. InnerSource é um passo em direção a todas essas conquistas.

Sobre o autor

Andy Oram é editor da O'Reilly Media. Funcionário da empresa desde 1992, Andy atualmente se especializa em tecnologias de código aberto e engenharia de software. Seu trabalho para O'Reilly inclui os primeiros livros já lançados por uma editora norte-americana no Linux, o título Peer-to-Peer de 2001 e o best-seller de 2007 Beautiful Code.