PROJETO*: práticas ágeis em um projeto de software livre

Autores

[1]Instituições

Abstract. *English abstract*

Resumo. *Embora surgidas em contextos diferentes, e com motivações diferentes, há vários aspectos em comum entre as práticas ágeis e as práticas tradicionalmente adotadas em projetos de software livre. Neste artigo apresentaremos o projeto PROJETO como um estudo de caso sobre a aplicabilidade de práticas ágeis em um projeto de software livre não-comercial. Iremos adotar como referência as práticas da Programação Extrema (XP) e iremos explorar quais são as práticas da XP utilizadas e as não utilizadas no projeto, assim como as motivações para a utilização ou não de cada prática. Com isso, nossa contribuição é a apresentação de evidências empíricas de um caso concreto sobre a relação entre práticas ágeis e projetos de software livre.*

1. Introdução

Short paper: de 6 a 8 pgs; **Full paper: de 12 a 16 pgs**

Obs: o texto ainda está bem em modo rascunho pq estou tentando primeiro jogar todas as ideias aqui.

Embora surgidas em contextos diferentes, e com motivações diferentes, há vários aspectos em comum entre as práticas ágeis e as práticas tradicionalmente adotadas em projetos de software livre (SL). Essa relação entre software livre e práticas ágeis tem sido estudada por pesquisadores [Corbucci and Goldman 2010, During 2006].

Logo no prefácio da segunda edição do livro "Programação Extrema" [Beck 2004], Erich Gamma já aborda as práticas ágeis utilizadas em um projeto de software livre, o projeto Eclipse. Essa menção sugere uma proximidade entre esses dois mundos. No entanto, ainda nesse prefácio, é declarado que nem todas as práticas da XP são usadas no projeto Eclipse. Seriam as práticas da XP não utilizadas nesse caso inadequadas para um projeto SL?

Os métodos ágeis nasceram com um foco bem grande na estruturação de pequenos times co-localizados que se dedicam em tempo integral a um projeto. Já projetos SL, por outro lado, podem ter dezenas ou centenas de colaboradores geograficamente distribuídos e que colaboram de forma voluntária. Essas diferenças põe em cheque a aplicabilidade das práticas ágeis em projetos SL.

Para colaborar com a discussão sobre a aplicabilidade das práticas ágeis em projetos SL, neste artigo apresentaremos um projeto de software livre, o PROJETO, como

*Devido ao sistema de *blind review* do WSL 2015, o nome do projeto utilizado neste estudo de caso será rasurado com "PROJETO". Outros contextos que possibilitem a fácil identificação do projeto serão rasurados com "@@@" .

estudo de caso. Iremos adotar como referência as práticas da Programação Extrema como representantes das práticas ágeis, e iremos explorar:

1) Quais as práticas da XP adotadas no PROJETO e os benefícios que elas trazem no contexto específico de um projeto de software livre.

2) Quais as práticas ágeis não adotadas no PROJETO e o porquê isso acontece.

Nossa intenção é que essas perguntas sejam respondidas com alguma profundidade, uma vez que os autores deste artigo são também desenvolvedores do PROJETO.

Com este artigo, esperamos apresentar evidências empíricas sobre a relação entre práticas ágeis e projetos de software livre. Em nosso contexto, as análises, interpretações e conclusões se restrigem a um grupo restrito de projetos de SL: aqueles que não possuem um apoio financeiro regular de alguma instituição e são, portanto, mantidos principalmente por voluntários.

Esperamos que as discussões aqui apresentadas possam ser de utilidade principalmente para comunidades de SL que estejam adotando ou discutindo a adoção de práticas ágeis. Além das comunidade de SL, acreditamos que possam se benefeciar da discussão desenvolvedores atuando em equipes geograficamente distribuídas, mesmo em projetos não SL.

2. Sobre o projeto PROJETO

3. Métodos ágeis e a Programação Extrema

Intro sobre ágil...

Intro sobre XP... valores... *Comunicação*, *Coragem*, *Feedback*, *Respeito*, e *Simplicidade*.

Apresentaremos agora brevemente as práticas da XP descritas na segunda edição do livro *Extreme Programming Explained* [Beck 2004]:

Sentar-se junto: a equipe deve trabalhar em um mesmo espaço de trabalho para promover a comunicação. Espaços abertos são preferíveis aos cubículos.

Equipe integral: a equipe deve incluir pessoas de forma a conter todas as habilidades necessárias para o sucesso do projeto. Exemplos de perfis possivelmente necessários: desenvolvedores, analistas de negócios, administradores de sistemas, testadores, etc. Todos devem estar comprometidos com o sucesso do projeto.

Ambiente informativo: algum interessado deve ser capaz de entrar no ambiente de trabalho e obter em poucos segundos uma ideia do andamento do projeto. Com mais uma olhada, ele deve ser capaz de captar possíveis problemas que estejam acontecendo.

Trabalho energizado: trabalhe apenas o tempo em que conseguir manter um ritmo produtivo. O desenvolvimento de software é um trabalho criativo, por isso trabalhar mais horas não é o melhor caminho para a entrega de um produto melhor.

Programação em par: dois desenvolvedores trabalham juntos na mesma estação de trabalho. Com essa prática, o código (junto de seu *design* e testes) já nascem de forma muito mais criteriosa e, portanto, com maior qualidade.

Histórias: o cliente deve escrever pequenas histórias que ilustrem novas funcionalidades ou capacidades do sistema. O esforço de implementação dessas histórias é estimado pela equipe, e assim elas podem ser priorizadas pelo cliente.

Ciclo semanal: a equipe e o cliente devem realizar um planejamento semanal do que a equipe irá entregar na próxima semana. Isso promove o desenvolvimento incremental e acelera o ciclo de *feedback*.

Ciclo trimestral: além do planejamento semanal, um planejamento de maior prazo é realizado trimestralmente, no qual é possível haver mais reflexão sobre o projeto, além de discussões sobre os gargalos enfrentados.

Folga: deixe sempre uma folga no planejamento. Se sobrar tempo, é possível entregar algo a mais ou utilizar o tempo extra para aprendizado. Mas se comprometer com o que não pode ser entregue pode gerar muitos desgastes nos relacionamentos com o cliente e com a organização.

***Build* de dez minutos:** o sistema deve ser construído (compilado, testado e empacotado) em até 10 minutos. Um tempo maior é um incentivo para que o desenvolvedor não execute o *build* com frequência, o que reduz o *feedback* e deteriora a qualidade do produto.

Integração contínua: suas mudanças no código devem ser continuamente integradas com o trabalho de outros colegas. Quanto mais tempo se espera para integrar diferentes fluxos de trabalho, maior será o esforço e a dificuldade em se completar com sucesso a integração.

Desenvolvimento Orientado a Testes: escreva primeiro um teste falhando para especificar o comportamento do próximo trecho de código a ser escrito. Assim o desenvolvedor terá um foco maior em escrever código que realmente importa. Além disso, a escrita de testes automatizados fornece constante *feedback* sobre a qualidade do *design* do código.

Design incremental: investimento em um bom *design* deve ser realizado constantemente ao longo do desenvolvimento do sistema. Um bom *design* promoverá a capacidade de introduzir mudanças no sistema posteriormente, o que é o grande objetivo da XP (*abrace a mudança*). A melhoria contínua do *design* é preferível a uma grande fase inicial de *design*.

4. Trabalhos relacionados

Ahalt et al., por exemplo, apresenta um processo de desenvolvimento de software livre no contexto de pesquisas científicas com a adoção de práticas ágeis [Ahalt et al. 2014]. Já Corbucci e Goldman aplicaram questionários nas comunidades de software livre e de métodos ágeis para descobrir que ferramentas (alerta de falha em *builds*, *issue trackers*, etc.) cada comunidade valorizava mais. Já Tsirakidis et al., tentam encontrar fatores de sucesso e de falha em projetos de software livre com práticas ágeis [Tsirakidis et al. 2009].

5. Práticas da XP e suas aplicabilidades em um projeto de software livre

Nesta seção descreveremos a relação entre as práticas da XP e as práticas adotadas no PROJETO. Quando a prática for aplicada, se possível, tentaremos sugerir uma explicação de como essa prática ajuda e entregar valor no contexto de um projeto de software livre. Quando a prática não for aplicada, tentaremos encontrar as razões e se há práticas alternativas no projeto que procurem atingir o mesmo objetivo que a prática da XP.

Ao estabelecer essa relação, percebemos que adoção ou não, bem como eventuais adaptações, das práticas da XP depende do *modo de desenvolvimento*. Um projeto SL pode receber contribuições em contextos variados. Cada um desses contexto pode impor

um modo de desenvolvimento específico, que apresentará forçar favoráveis ou contrárias para a adoção de determinadas práticas. No projeto do PROJETO, podemos encontrar os seguintes modos de desenvolvimento:

Desenvolvimento distribuído com contribuição voluntária: esse é o modo clássico de desenvolvimento de software livre. Colaboradores geograficamente dispersos interagem virtualmente por meio de lista de e-mail e *issue tracker*. As contribuições são voluntárias, então normalmente não se pode controlar o momento e o conteúdo das contribuições. Embora a comunidade possa eleger metas e prioridades, cada contribuidor pode colaborar com o problema que lhe for mais interessante. Este é o modo predominante no desenvolvimento do PROJETO.

Desenvolvimento com equipe co-localizada em *hackathons*: eventualmente membros da comunidade se encontram por alguns dias para um esforço intensivo de desenvolvimento. Normalmente esse esforço se insere no contexto de uma competição, para a qual os membros da comunidade estipulam metas a serem atingidas nesse curto espaço de tempo (normalmente de 1 a 4). O PROJETO já participou de pelo menos três *hackathons* competitivas promovidas por organizações externas ao projeto.

Desenvolvimento feito por estudantes no contexto de uma disciplina: em cursos superiores relacionados ao desenvolvimento de software é comum a existência de disciplinas que demandem o desenvolvimento de projetos de software feitos pelos estudantes. Algumas universidades já têm incentivados aos alunos que desenvolvam esses projetos criando SL ou contribuindo com projetos SL já existentes. Em particular, há 4 semestres o PROJETO tem recebido contribuições de alunos de Engenharia de Software da Universidade @@@. Nesse contexto, é possível impor uma disciplina maior no desenvolvimento, exigindo-se o uso de certas práticas dos estudantes, bem como impondo uma revisão de código mais criteriosa. Por outro lado, os estudantes são livres para contribuir com os aspectos que desejarem.

O primeiro modo é que normalmente se entende como o modo clássico de desenvolvimento de software livre. Entendemos que o segundo modo, embora bem restrito a momentos específicos, seja bem comum em projetos de SL [During 2006]. Por fim, o terceiro modo pode não ser tão comum em projetos de SL, mas há alguns exemplos, especialmente para softwares nascidos no contexto de pesquisas científicas [Santos et al. 2012].

5.1. Sentar-se junto

O compartilhamento de um espaço físico comum entre os membros da equipe é a primeira grande diferença entre equipes ágeis, formadas por profissionais que trabalham juntos em uma organização, e equipes de projetos SL, tipicamente distribuídas.

Projetos SL são tipicamente compostos por pessoas distribuídas geograficamente, que muitas vezes nem se conhecem pessoalmente. Essa situação típica é o caso do PROJETO. Nesse contexto, é inviável seguir a prática de compartilhamento de um espaço físico de trabalho.

No entanto, é preciso considerar que o maior objetivo do compartilhamento do ambiente de trabalho é a promoção da comunicação. No PROJETO, a ausência da colocalidade é compensada pela comunicação via lista de e-mail e *issue tracker*. Em algumas situações mais específicas, *hangouts* são feitos entre os desenvolvedores. E há

também ocasionalmente a promoção de *hackathons*, onde o trabalho co-localizado é exercido intensamente por um curto período de tempo.

A utilização de listas de e-mail e *issue tracks* é bem comum também em outros projetos SL [Corbucci and Goldman 2010]. Também encontramos evidências sobre a promoção de *hackathons* periódicos em outros projetos [During 2006]. Não temos notícias ou referências sobre a utilização de *hangouts* em outros projetos SL.

5.2. Equipe integral

Acreditamos que essa prática a princípio não se aplica aos projetos SL, uma vez que todos os contribuidores de um projeto SL formam a "equipe". Assim, a adesão a essa prática em um projeto SL é compulsória.

Em organizações, a orientação para uma equipe integral faz sentido pois muitas vezes há uma equipe comprometida com o produto, mas algumas atividades são terceirizadas para outras equipes da organização, tais como atividades de testes ou implantação. Em projetos SL a princípio não há o conceito de "outras pessoas da organização fora da equipe".

No entanto, uma interpretação mais específica pode ser feita no contexto de grupos de alunos trabalhando no PROJETO para uma disciplina: o sucesso da equipe no contexto da disciplina não pode depender de membros do projeto que não estejam no grupo da disciplina. O mesmo vale para *hackathons*: atingir um objetivo em um *hackathon* não pode depender de contribuidores não presentes no *hackathon*. Assim, no contexto de uma disciplina ou de um *hackathon*, a equipe integral é uma prática importante a ser seguida em projetos SL.

5.3. Ambiente informativo

Em um projeto Software Livre não faz sentido falar de um ambiente *físico* informativo. Mas no PROJETO, temos algumas informações que poderiam formar um espaço *virtual* informativo, tais como: estado da *build* executada pela integração contínua, quantidade de visitas à aplicação no período, número de curtidas no Facebook no período, quantidade de linhas de código entregues no período, quantidade de issues fechadas no período, métricas de qualidade de código (atualmente em andamento). No contexto do PROJETO, o problema é que todas essas informações estão dispersas em páginas diferentes. Mas se fossem integradas em um único espaço virtual, formariam um ambiente informativo equivalente à prática da XP que poderia trazer novos benefícios ao projeto. Assim acreditamos que essa prática da XP é aplicável e desejável a projetos SL.

5.4. Trabalho energizado

No modo de desenvolvimento distribuído, cada colaborador age voluntariamente e, portanto, é autônomo para decidir a distribuição de seu esforço ao longo do tempo. Um colaborador pode individualmente seguir a prática de trabalho energizado e trabalhar apenas quando se sentir produtivo, mas acreditamos que não faz sentido falar dessa prática como algo adotado pelo projeto.

No caso dos *hackathons*, a situação é extremamente oposta. Como são raros momentos com apenas poucos dias de duração, é comum que os participantes trabalhem durante todo o dia incluindo até a madrugada. É a tentativa de aproveitar ao máximo

possível um momento de esforço intensivo e de possibilidade de interação presencial com os outros membros do projeto.

No modo de disciplina a lógica é similar ao *hackathon*, mesmo que mesmo dramática. Os alunos em geral possuem o período da aula para exercer suas atividades. Mesmo que apliquem algum esforço extra-classe, em geral é apenas na sala de aula que todos os membros estão co-localizados, fazendo com que esse momento deva ser aproveitado ao máximo possível, mesmo que eventualmente em alguns dias nem todos os membros estejam tão "energizados".

5.5. Programação em par

Devido a distribuição geográfica e às limitações tecnológicas ainda existentes, a programação pareada é rara no modo distribuído. No entanto, é comum que um desenvolvedor mais experiente no projeto revise o código entregue por colaboradores mais ocasionais. Essa revisão é feita de forma mais intensa e criteriosa no modo de disciplina. Já no modo de *hackathon* a programação normalmente é adotada, possibilitando a troca de conhecimentos e ideias entre colaboradores que raramente se encontram presencialmente.

5.6. Histórias

No PROJETO, não temos colaboradores atuando com o papel de cliente. Em parte isso se deve ao uso geral da aplicação: qualquer cidadão é um potencial usuário. Por outro lado, essa não é uma limitação inerente a projetos de SL, e mesmo em nosso caso acreditamos que o projeto teria muito a ganhar se trabalhasse de uma forma mais voltada a demandas/sugestões de usuários mais ativos na comunidade. Para amenizar essa situação, nos eventos comunitários dos quais participamos, procuramos ouvir o *feedback* dos usuários do PROJETO. Uma importante funcionalidade do PROJETO, que é @@@, surgiu de sugestões de diversos usuários que tiveram o mesmo sentimento de que tal funcionalidade era desejada.

Ao mesmo tempo, embora não sejam definidas por usuários, no modo *hackathon* e de disciplina, o time elege funcionalidades a serem implementadas em um dados período de tempo, o que se aproxima da prática da seleção de histórias. Em ambos os casos, desenvolvedores mais experientes no projeto ajudam o time a equacionar o compromisso entre valor e dificuldade de implementação da história.

5.7. Ciclo semanal

No modo de desenvolvimento distribuído com contribuições voluntárias tal prática não é imposta, já que nesse modo as contribuições normalmente ocorrem intervalos maior do que uma semana. Já no modo de *hackathon*, o evento todo constitui um único ciclo curto. Por fim, mesmo no modo de disciplina, uma semana é um período curto demais, já que a semana conta com apenas 8 horas de desenvolvimento.

1a edição: ciclos curtos; SL: release early

5.8. Ciclo trimestral

Embora informal, o PROJETO apresenta períodos no qual o esforço da comunidade se concentra majoritariamente em determinadas funcionalidades. Embora não seja preciso, acreditamos que esse período em que se eleja prioridade que se reflita mais sobre os rumos

do projeto seja semestral. Um fator importante para essa reflexão semestral é a discussão das potenciais contribuições que os alunos das disciplinas possam entregar ao projeto, já que as disciplinas são semestrais.

5.9. Folga

Acreditamos que esse princípio não se aplica ao modo de desenvolvimento distribuído com contribuições voluntárias. No entanto, pode ser aplicado no planejamento de *hackathons* e das disciplinas. No entanto, na prática acreditamos que normalmente "folga" não seja um valor predominante na execução de *hackathons*. Quanto às contribuições nas disciplinas, TODO...

5.10. *Build* de dez minutos

O PROJETO possui uma bateria de testes automatizados. O maior foco é nos testes de unidade, que são os que fazem testes das classes e métodos do código (isolando totalmente ou não as unidades). O projeto possui ainda alguns poucos testes de integração com serviços externos (@@@), mas nenhum teste de usuário, no qual a interface seria manipulada de forma automatizada sob a perspectiva do usuário. A presença de poucos testes de integração e nenhum teste de aceitação faz com que a *build* seja executada em menos de 2 minutos.

Acreditamos que independente do modo de desenvolvimento do projeto de SL, esta prática seja importante e deva ser seguida por qualquer projeto de software no qual o uso de testes automatizados se aplique.

5.11. Integração contínua

A cada entrega realizada no repositório, a bateria de testes automatizados é executada. Caso a *build* falhe, um e-mail é enviado aos membros do projeto, dando visibilidade imediata do ocorrido.

Acreditamos que a existência de uma bateria de testes automatizados traga um valor a mais para projetos de SL dependentes de voluntários, quando comparado a projetos de software em geral. Isso porque colaboradores dedicam seu pouco tempo livre ao desenvolvimento do projeto. A existência de uma bateria de testes dá confiança ao colaborador de que ele não está inserindo defeitos no projeto, dispensando-o do esforço de ter uma compreensão de todos os aspectos do sistema, podendo se focar apenas na contribuição de seu interesse.

No PROJETO tomamos um passo além da integração contínua e estamos investindo num fluxo de entrega contínua, no qual a integração contínua dispara a implantação da nova versão da aplicação caso a bateria de testes tenha sucesso. Novamente, no contexto de projetos de SL dependentes de voluntários isso tem um valor a mais, uma vez que se alivia o esforço de gerenciamento da infra. Sem a implantação automatizada, toda contribuição entregue por um colaborador só vira valor para o usuário depois que outro colaborador com acesso à infra atualize o sistema em produção. Com a implantação contínua, o código entregue por qualquer colaborador vira valor imediatamente valor para o usuário, acelerando o ciclo de *feedback*, o que é importante, pois quanto mais cedo um voluntário tenha o *feedback* de suas alterações, maiores são as chances de que ele esteja disposto a refinar a solução baseando-se no *feedback* obtido.

Acreditamos que a implementação do fluxo de entrega contínua seja uma estratégia crítica, pois atualmente um esforço em uma *hackathon* não pode ser publicizado sem que um dos participantes com domínio da infra (atualmente são dois no PROJETO) estejam presentes no evento. No modo da disciplina isso também já é estratégico, já que os alunos não possuem acesso à infra de produção.

5.12. Desenvolvimento Orientado a Testes

No modo de desenvolvimento no qual voluntários contribuem distribuidamente não é possível impor uma disciplina de "testes antes", como advoga o TDD, já que o código é entregue com os testes, não sendo possível a verificação do que veio antes. Mesmo a entrega de código com testes é uma exigência que em alguns contextos poderia inibir a colaboração. Portanto no PROJETO a entrega de testes não é exigida. Por outro lado, em alguns contextos acontece de que desenvolvedores refatorem e entreguem testes para código já entregue por outros colaboradores.

No modo *hackathon* a pressa em se gerar resultados tendo em vistas uma possível premiação é um grande fator para a geração de dívida técnica. Ou seja, nessa situação é comum que muito código seja produzido sem testes automatizados. Mas assim como no modo distribuído, em alguns contextos parte da dívida técnica é paga depois do *hackathon*, com a produção de testes automatizados.

Por fim, o modo de disciplina é o mais favorável à aplicação de uma maior disciplina na entrega de código com testes automatizados. Nesse contexto, é possível colocar como objetivo da disciplina a prática de TDD, por exemplo. No entanto, mesmo que o professor possa impor essas linhas, a comunidade não consegue obrigar que os alunos entreguem o código com testes, embora normalmente isso não aconteça.

Por fim, mesmo com todas essas dificuldades, vale ressaltar que na última medição, o PROJETO apresentou um índice de cobertura de testes de 64%, o que consideramos um resultado satisfatório.

5.13. Design incremental

Em todos os modos de desenvolvimento há um esforço por parte dos desenvolvedores mais experientes em se manter um *design* limpo e elegante para a aplicação. Isso é feito com refatorações sobre código entregue por outros colaboradores e com aconselhamento a esses outros colaboradores sobre o que fazer.

Um desafio em particular dentro de uma estratégia para um *design* limpo que evolua incrementalmente é o desincentivo à entrega de código que não entrega valor ao usuário. Consideramos que isso acontece quando um colaborador se antecipa em entregar código que só terá valor quando uma outra funcionalidade for implementada. Se não há uma perspectiva próxima para a implementação dessa nova funcionalidade, o código entregue pode acabar não tendo uma utilidade. Isso trás dois grandes problemas no contexto de um projeto SL dependente de colaborações voluntárias: 1) o esforço gasto pelo colaborador poderia ter sido canalizado para algo de mais valor, que ajudaria de fato na melhoria do software; e 2) embora o código não esteja sendo utilizado, é preciso mantê-lo e continuamente garantir que o mesmo não seja quebrado, o que eventualmente pode demandar um esforço extra de outros colaboradores que poderia não ser necessário.

Acreditamos que essas considerações se apliquem a qualquer projeto de SL dependente de voluntários.

6. Ameaças a validade

Viés: alunos UnB podem ser "obrigados" a seguirem algumas práticas ágeis. Por outro lado, a contribuição se caracteriza bem no recorte de projetos SL, já que os alunos decidem as funcionalidades que entregaram (embora haja recomendações, essas recomendações acontecem para qualquer contribuidor).

Estudo de caso vs análise por amostragem vs simulação [Turnu et al. 2004].

Como em qualquer estudo de caso, há o viés de que estamos observando apenas um caso particular, mas em compensação podemos fazer uma análise qualitativa com mais profundidade para entender, nesse contexto, porque tais práticas específicas são utilizadas ou não.

a referência foi a XP, mas se fosse scrum, p ex. resultado seria diferente

7. Conclusão

Logo no prefácio da segunda edição do livro "Programação Extrema" [Beck 2004], Erich Gamma já aborda as práticas ágeis utilizadas em um projeto de software livre, o projeto Eclipse. No entanto, ele mesmo já declara que nem todas as práticas da XP são usadas. Isso já evidencia como tais práticas devem ser adaptadas e ponderadas dentro de cada contexto. E diferentemente do software produzido por times profissionais que trabalham em tempo integral e de forma co-localizada, projetos de software livre possuem peculiaridades que destoam desse cenário para o qual os métodos ágeis foram pensados.

Práticas técnias mais propensas a serem adotadas, práticas gerenciais menos propensas... e a práticas gerenciais menos propensas possivelmente pq: time profissional comprometido com meta vs voluntariarismo mais oportunístico...

Também constatamos que as práticas gerencias são as q mais variam com o modo de desenvolvimento, enquanto que as práticas técnicas tem maior potencial de permanecerem uniforme pelos modos.

Referências

Ahalt, S., Band, L., Christopherson, L., Idaszak, R., Lenhardt, C., Minsker, B., Palmer, M., Shelley, M., Tiemann, M., and Zimmerman, A. (2014). Water science software institute: Agile and open source scientific software development. *Computing in Science & Engineering*, 16(3):18–26.

Beck, K. (2004). *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2nd edition.

Corbucci, H. and Goldman, A. (2010). Open source and agile methods: Two worlds closer than it seems. In *Agile Processes in Software Engineering and Extreme Programming*, pages 383–384. Springer.

During, B. (2006). Trouble in paradise: the open source project pypy, eu-funding and agile practices. In *Agile Conference, 2006*, pages 11–pp. IEEE.

Santos, V. A., Goldman, A., and Santos, C. D. (2012). Uncovering steady advances for an extreme programming course. *CLEI Electronic Journal*, 15(1):2–2.

Tsirakidis, P., Kobler, F., and Krcmar, H. (2009). Identification of success and failure factors of two agile software development teams in an open source organization. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pages 295–296. IEEE.

Turnu, I., Melis, M., Cau, A., Marchesi, M., and Setzu, A. (2004). Introducing TDD on a free libre open source software project: a simulation experiment. In *Proceedings of the 2004 workshop on Quantitative techniques for software agile process*, pages 59–65. ACM.