@@@*: 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 @@@ 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

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. Essa relação já tem sido estudada por pesquisadores, principalmente no contexto de projetos livres sem o patrocínio de alguma organização.

Neste artigo apresentaremos um projeto de software livre, o @@@, como estudo de caso sobre a aplicabilidade das práticas ágeis em um projeto de software livre não-comercial. Iremos adotar como referência as práticas da Programação Extrema e iremos explorar:

1) Quais as práticas ágeis adotadas no Radar e os benefícios que elas trazem no contexto de um projeto de software livre.

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

Com isso, esperamos apresentar evidências empíricas sobre a relação entre práticas ágeis e projetos de software livre. 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.

Método: autores deste artigo são desenvolvedores do projeto, então podem conseguem enxergar bem as motivação e desmotivações envolvidas na adoção das práticas.

Contexto: recorte do tipo de projeto software livre

Além de relatar, este trabalho pode ajudar a sugerir práticas ágeis ao @@@ e outros projetos de software livre que ainda não estejam sendo utilziadas....

*O nome do projeto utilizado neste estudo de caso, assim como qualquer contexto que possibilite sua fácil identificação, serão ocultos com a rasura "@@@" devido ao sistema de *blind review* do WSL 2015.

Aplicações das lições não só pra SL, mas para projetos distribuídos em geral, já q maior dif entre ágil e livre está na distribuição.

2. Sobre o 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 2000]:

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: práticas ágeis em projetos software livre

Intro do livro XP, por Erich Gamma, já fala as práticas ágeis que o projeto Eclipse utiliza! Várias são listadas, mas ele fala que nem todas são usadas.

Citando galera [Ahalt et al. 2014, Turnu et al. 2004, 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 @@@. 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 @@@, 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 @@@.

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 @@@ 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 @@@ 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.

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 @@@. 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 @@@, a ausência da co-localidade 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 @@@ 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 @@@, 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 @@@, 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

5.5. Programação em par

5.6. Histórias

5.7. Ciclo semanal

5.8. Ciclo trimestral

5.9. Folga

5.10. *Build* de dez minutos

5.11. Integração contínua

5.12. Desenvolvimento Orientado a Testes

5.13. Design incremental

5.14. Jogo de Planejamento

5.15. Fases pequenas

5.16. Metáfora

5.17. Design Simples

5.18. Testes de Aceitação

5.19. Semana de 40 horas

5.20. Propriedade Coletiva

5.21. Padronização do Código

5.22. Refatoração

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.

7. Conclusão

Diferença: time profissional comprometido com meta vs voluntariarismo mais oportunístico...

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. (2000). *Extreme programming explained: embrace change*. Addison-Wesley Professional.

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.

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.