Uma Abordagem Empírica para o Tratamento de Bugs em Ambientes Ágeis

Andreia Matos dos Santos [1,2], Börje F. Karlsson [2], André M. Cavalcante[1,2]

[1] Programa de Engenharia de Produção, Universidade Federal do Amazonas (UFAM)
Av. General Rodrigo Octávio Jordão Ramos, 3000, 69077-000, Manaus-AM, Brazil

[2] Instituto Nokia de Tecnologia (INdT)
Av. Torquato Tapajós, 7200, 69093-415, Manaus-AM, Brasil

{andreia.santos, borje.karlsson, andre.cavalcante}@indt.org.br

Abstract. *For some time, software development companies are looking for new methodologies in order to optimize the development process, agile methodologies are becoming increasingly popular and testing of software is undergoing changes to incorporate this new reality. In this paper a study is reported in a research institute to discuss the practical work on defect management for agile teams. Through the practical application of different approaches, we discuss what the practice is most appropriate for the treatment of defects in projects using Agile with Scrum, relevant items such as rapid dissemination of the defect, prioritizing solutions, aid defect management tools, among other serve as the basis for the study.*

Resumo. *Há algum tempo as empresas desenvolvedoras de software vem buscando metodologias novas com o objetivo de otimizar os processos de desenvolvimento, nesse sentido as metodologias ágeis estão se tornando cada vez mais populares e o teste de software vem passando por adaptações para se integrar a esta nova realidade. Neste artigo será relatado um estudo em um instituto de pesquisa para discutir o trabalho prático sobre tratamento e gerenciamento de defeitos pelos times ágeis. Através da aplicação prática de diferentes abordagens, discutimos qual a prática mais adequada para o tratamento de bugs em projetos utilizando a metodologia ágil com Scrum, ítens relevantes como divulgação rápida do defeito, priorização de solução, auxílio de ferramentas de gerenciamento de defeitos, entre outros servirão de base para o estudo.*

1. Introdução

O nível de demanda dos consumidores por produtos de software é cada vez mais elevado, portanto, fatores como entregas em menores tempos e qualidade do produto ou serviço se tornam cruciais. Problemas como o alto custo, alta complexidade, dificuldade de manutenção, e uma disparidade entre as necessidades dos usuários e do produto a ser desenvolvido, tornam-se cada vez mais evidentes nos processos de desenvolvimento de software [Sommerville 2004].

Para atenuar esta situação, diferentes metodologias têm sido propostas e implementadas em empresas de desenvolvimento de software. Diferentemente das

metodologias tradicionais, as metodologias chamadas de ágeis (em ampla adoção) possuem princípios diferentes e são conhecidas por reduzir a burocracia associada a atividades de desenvolvimento de produto [Fowler 2001].

O teste de software é um componente central no desenvolvimento de software, e em particular de metodologias ágeis, onde são considerados práticas fundamentais. Técnicas como *test-driven development* (TDD) [Beck 2002], testes de unidade [Meszaros 2007], refatoração [Fowler 1999], e até mesmo guias de boas práticas para trabalhar com código legado [Feathers 2004] são alguns exemplos. Muitos são os desafios do teste diante da nova realidade uma vez que as pequenas entregas em curto espaço de tempo, as constantes mudanças de requisitos e pouca documentação são impactantes na garantia da qualidade do que está sendo desenvolvido, além do gerenciamento dos defeitos encontrados.

Ao contrário das métodologias tradicionais nas quais os testes ocorrem em uma fase posterior dentro do processo de desenvolvimento, testes ágeis devem ocorrer de forma frequente e desde o início, procurando detectar defeitos o mais cedo possível dentro de ciclos de desenvolvimentoiterativos e curtos, com um constante *feedback* do cliente. Em projetos ágeis, o código é considerado completo apenas se passar pelos testes e os testes são considerados parte da especificação dos testes.

O estudo reportado neste trabalho visa fornecer relatos de experiências em desenvolvimento ágil de software em um instituto de pesquisa e desenvolvimento, porém com foco no gerenciamento de *bugs* em projetos dentro da realidade de um processo de desenvolvimento ágil segundo Scrum. Para atingir esse objetivo, é feita uma análise sobre dos métodos utilizados, e das experiências de outros autores, bem como a revisão de alguns conceitos importantes para o gerenciamento de defeitos. As experiências de desenvolvimento em si e os passos tomados pelos times de projeto são descritos em mais detalhe em [Santos et al. 2011].

Este artigo está organizado da seguinte forma: A Seção 2 apresenta uma breve descrição sobre conceitos importantes para os testes ágeis no contexto de tratamento de bugs. A Seção 3 descreve o estudo de caso que foram utilizados para fornecer esse relato de experiência. Além disso, esta seção destina-se a identificar alguns importantes problemas em lidar com o gerenciamento de defeitos em ambientes ágeis e como, finalmente, conclusões são apresentadas na Seção 4.

2. Trabalhos relacionados

2.1 Qualidade de Software e Defeitos

Em diversos ambientes onde se desenvolve software a mera menção da palavra bug equivale a acionar um alarme, o que tende a causar “pânico”, isto é, existe um sentimento preponderante de que qualidade de software e bugs são coisas opostas e incompatíveis.

No entanto, a descoberta de *bugs* pode também ser vista como essencial. Em 1979, Myers já apresentava o conceito de que quanto mais cedo se descobre e se corrige um erro, menor é o seu custo para o projeto [Myers 1979]. Dentre as fases de Especificação, Projeto, Construção, Teste e Produção, o custo de correção do bug

aumenta conforme avança pelo ciclo de desenvolvimento, crescendo numa progressão logarítmica (a Figura 1 mostra um exemplo).

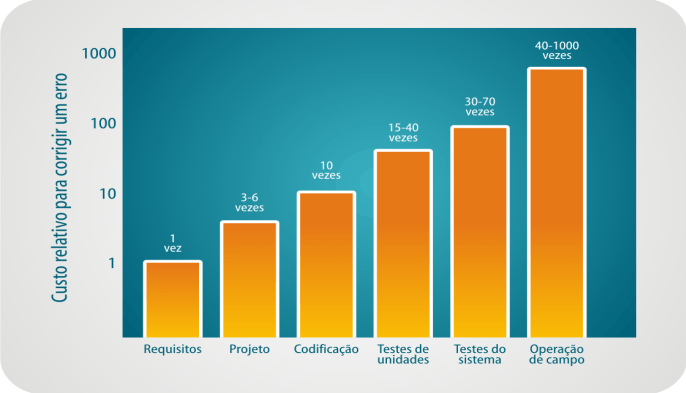

Figura 1. Exemplo da progressão de custo da correção de erros [Gritti 2010]

Apesar do custo médio de correção de um defeito ser muito mais baixo quando encontrado em fases iniciais do ciclo de vida de desenvolvimento de um software, apenas testes realizados durante a fase de codificação não garantem a inexistência de defeitos. Testes de mais alto nível, usualmente desempenhados por testadores (por exemplo: de integração e de sistema) podem detectam um outro tipo problemas, mas mesmo assim só cobrem cerca de 60% das possibilidades [Glass 2002].

Em parte para mitigar esses problemas, o papel do testador em um processo ágil é diferente. De acordo com Crispin e Gregory [2009], as principais atividades desempenhadas por um testador num projeto ágil são:

- Negociar qual o nível de qualidade esperado pelo cliente (não o seu padrão de qualidade, mas o padrão que cliente desejar e estiver disposto a pagar);
- Clarificar estórias e esclarecer suposições;
- Prover estimativas para as atividades de desenvolvimento e testes;
- Garantir que os testes de aceitação verificam se o software foi construído conforme o nível de qualidade definido pelo cliente;
- Ajudar o time a automatizar os testes e a desenvolver código testável;
- Prover *feedback* contínuo para manter o projeto no rumo.

2.2 Testes Exploratórios

Teste exploratório de software é uma abordagem poderosa de testes. Em algumas situações, pode ser várias ordens de grandeza mais produtivo do que teste com scripts. Dificilmente se encontra um testador que não fez, pelo menos inconscientemente, testes exploratórios em um momento ou outro [Bach 2010].

Uma poderosa abordagem de testes para determinados tipos de software e projeto é o teste exploratório, que é definido como sendo, simultaneamente, aprendizagem, *design d*e testes e execução de testes [Bach 2003]; isso significa que os testes não são definidos antecipadamente em um plano de testes, mas que são dinamicamente projetados, executados e modificados. Por isso, menos preparação (em

relação a conhecimento dos requisitos e design prévio dos casos de testes) é exigida, e importantes defeitos são encontrados rapidamente, tornando-se mais estimulante intelectualmente para o testador do que a execução de testes de roteiro [Santamaria 2007].

Ter conhecimento o mais rápido possível de bugs é o ideal em qualquer tipo de metodologia de desenvolvimento, mas especialmente nas metodologias ágeis isso é imprescindivel.

2.3 Priorização dos Defeitos

No Scrum a equipe de teste deve estar envolvida com o Product Owner (P.O.) e com todo o time para ajudar a priorizar defeitos de forma rápida e eficiente. O sistema de exposição da priorização dos defeitos pode ser feito de maneira física, onde estes são colocados de uma maneira visual em um quadro ou ainda em uma ferramenta de *bug tracking*, onde os defeitos recebem uma qualificação de prioridade em acordo com todo o time [Kealy e Beeson 2008].

É importante notar que durante a priorização dos defeitos as seguintes perguntas devem ser respondidas:

- Qual a repetibilidade do defeito na aplicação?
- Quais as categorias que a maioria dos defeitos se encaixa?
- Quais defeitos precisamos corrigir em primeiro lugar?
- Qual é a área de funcionalidade que cada defeito afeta a aplicação?

Durante a sessão de planejamento de cada iteração, o time juntamente com o PO devem tentar encontrar uma resposta rápida às perguntas. Caso a maioria dos defeitos encontrados possuírem maior probabilidade de ocorrência e o impacto seja grande para o usuário, claramente a aplicação está na necessidade de correções desses bugs o mais rápido possível. Se o maioria dos defeitos encontrados forem menos provável de ocorrer e com impacto mínimo para o usuário, então o aplicativo está com qualidade boa. Após este exercício, ficará claro para o PO quais defeitos possuêm maior prioridade de correção, auxiliando o time a tomar decisões no âmbito de planejamento.

2.4 Definição de Pronto e Feito no Scrum

De acordo com Sutherland [2009] o conceito de “Pronto” são requisitos possíveis de serem implementados e “Feito” é o requisito desenvolvido conforme os critérios de aceitação do Product Owner. O “Feito” seria a coluna de “Done” de um quadro de Kanban (também chamado de *Taskboard*), que é definido entre a equipe e o Product Owner.

Para aumentar a velocidade da equipe com os requisitos “Prontos”, os mesmos são escritos pelo Product Owner com o auxilio do Scrum Master, eles juntos realizam um trabalho de preparação do material da próxima Sprint e quando este material chega ao planejamento da Sprint ele possui um nível de maturidade maior, para que a equipe possa realizar a analise de sistemas do mesmo e o seu planejamento de execução.

Os conceitos de Pronto e Feito são simples de entender, mas difíceis de ser "implementados", neste sentido o segredo está no balanceamento apropriado entre o planejamento e a execução de atividades [Sutherland 2009].

3. Estudo de Caso

Uma série de projetos desenvolvidos no Instituto Nokia de Tecnologia (INdT) durante os dois últimos anos e meio, todos usando o Scrum como arcabouço de gerência de projeto, fornece os dados utilizados neste estudo; sendo um deles utilizado como base para a aprsentação deste trabalho. Este projeto é brevemente descrito abaixo, uma vez que, devido a sua complexidade e tamanho, possibilitou a utilização de diferentes abordagens no tratamento de bugs. Apesar disto, as lições aprendidas e os ajustes realizados em outros projetos também forneceram informações sobre questões diferentes que servira, como entrada para o melhor entendimento das dificuldades encontradas e as possíveis soluções do problema.

3.1 Análise do problema

Inicialmente foram realizadas observações em diversos projetos executados dentro do INdT, sobre a maneira como os bugs eram tratados nos sprints e levantamos os pontos positivos das abordagens e principalmente os negativos, conforme segue:

a) Projeto M: Os bugs encontrados não eram priorizados por todo o time, eram encontrados ao final do Sprint nem todos eram corrigidos por falta de tempo hábil. A responsabilidade em levantar os bugs era apenas do analista de teste que reportava os bugs na ferramenta de bugtracking. Neste caso, o PO não analisava a prioridade dos bugs, dava a estoria como concluída e a decisão de corrigir ou não era dos desenvolvedores.

- Vantagem: As estórias não dependiam de bugs corrigidos para serem consideradas pronta, o que trazia uma impress"ao de progresso na implementação das funcionalidades.
- Desvantagem: A correção dos bugs era postergada e a maioria não era corrigida, apenas os bugs *blocker* eram corrigidos.

b) Projeto N: Os bugs encontrados eram colocados em um bug backlog e transformados em uma estória com prioridade sobre as outras dentro do próximo sprint.

- Vantagem: Os bugs eram corrigidos antes que novas implementações fossem iniciadas.
- Desvantagem: Os bugs classificados como *trivial* ou *minor* que exigiam um pouco mais de tempo para resolver, nem sempre eram corrigidos, uma vez que o novo sprint (de 2 semanas) já estava iniciado e, portanto, novas funcionalidades deveriam ser desenvolvidas nesse tempo.

Após essa análise, a equipe de validação começou a se questionar em como Scrum recomenda que a equipe trate os bugs? Eles devem ser colocados no product backlog? Ou em uma lista de bugs separada? Se eles estão no backlog, o Product Owner deve definir as prioridades ou eles são automaticamente os itens mais importantes? Deve existir um sprint em separado para a correção de bugs?

3.2 Proposta para tratamento de bugs no Scrum

Com base nas experiências anteriores, no momento de início de um novo projeto uma nova abordagem foi proposta.

- Projeto D

De um ponto de vista geral o projeto D pode ser descrito como um programa que fornece uma solução para criação de publicidade para operações através de cupons digitais, de forma eficiente e mensurável. O programa é dividido em projetos com foco em desenvolvimento de aplicação para celular, serviço de *back end* e relatórios web para gerenciamento de informações.

Para o projeto D a melhoria na definição de pronto das estórias foi acordada com o PO e foram estabelecidas algumas metas de testes unitários e correção de bugs com o objetivo de evitar o máximo possível que bugs escapem do sprint. O foco da equipe era reduzir o tempo necessário para descobrir (e então corrigir) problemas. Além disso, descobertas precoces fariam com que fosse muito mais fácil corrigir já que o código ainda estava fresco na mente da equipe de desenvolvimento.

Neste sentido, se faz importante apresentar os requisitos que a equipe utiliza para categorizar os bugs segundo a prioridade na ferramenta de bug tracking Jira, antes de apresentarmos o processo definido.

Table 1. Nível de prioridade de *bugs*

Nível de Prioridade	Descrição
Blocker	Impede o funcionamento do sistema "quebrando" ou fechando o mesmo ou causa perda de dados, etc
Critical	Falhas graves, perda de dados, perda de memória.
Major	Grande perda de função da funcionalidade específica, sendo quo sistema continua funcionando.
Minor	Perda de função pequena, ou outro problema cuja solução seja fácil.
Trivial	Problemas "cosméticos", como palavras grafadas incorretamente ou texto desalinhados.
Enhancement	Solicitação de melhoria ou mudança

- Conceito de Pronto das estórias, segundo o P.O.:

- Devem ter sido implementadas todas as funcionalidades, conforme product backlog;
- Devem ter sido rodados testes funcionais;
- Testes unitários dos componentes do sistema principal em pelo menos 30% do código;
- Geração de relatório de execucão dos testes apresentada na reunião de Review para o time.

- Metas para correção dos Bugs:

- Bugs critical e major devem ser corrigidos dentro do Sprint;
- Bugs remanescente do Sprint corrente não devem ultrapassar 50% do total de bugs e ficam para ser corrigidos no próximo Sprint;
- Bugs minor ou trivial, foram vai registrar em bug backlog e resolvidos de acordo com a priorização de bug backlog acordado pelo PO e o time, e são considerados em uma nova história.

Bugs encontrados dentro do sprint são melhor tratados quando identificados imediatamente e relatados para toda a equipe nas *daily meetings* , se forem blockers ou criticals, por exemplo. Se isto não der certo, um cartão descrevendo o bug pode ser incluído no quadro de tarefas. Embora existam bugs que escapam do sprint é preferível adicioná-los no product backlog, deixando que o *product owner* os priorize. Muitas equipes ainda têm uma base de dados de bugs que precisam continuar usando [Cohn 2010].

Além do processo de tratamento de bug`s, baseado em priorização, como descrito anteriormente, a equipe de testes também adotou algumas tarefas padrões para todos os sprints de forma que aumentasse a agilidade na divulgação dos bugs, conforme segue:

1. Cobrar do time de desenvolvimento a execução dos teste unitários, o que minimizou a ocorrência de bugs blocker que impediam a continuidade dos testes;
2. Execução de testes exploratórios, feitos por analistas de testes experientes, assim que a versão de teste era liberada, com isso muitos bugs eram divulgados imediatamente e corrigidos pelos desenvolvedores.

É importante ressaltar que este processo de tratamento de bugs teve ajuda de todo o time, neste sentido o comprometimento de fazer pequenas entragas dentro do Sprint, onde cada estória era testada uma a uma, possibilitou reduzir a quantidade de bugs remanescentes entre os *Sprints*, o que aumentou a qualidade do sistema gradativamente conforme os *Sprints* eram fechados. A figura 2 representa o comprometimento de toda a equipe com as tarefas de teste e tratamento de bugs dentro das estórias, baseado nas funções de cada membro do time.

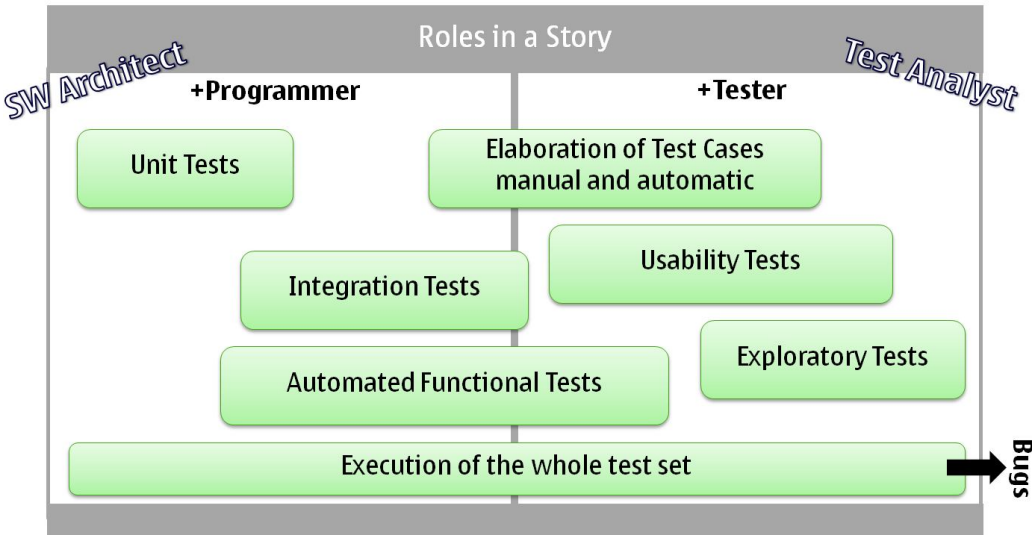

Figura 2. Papéis em um time ágil

Em concordâcia com o Agile Manifesto [Beck 2001]: "Software funcionando é a primeira medida do progresso". Pode-se consider como desonesta a atitutde de se passar adiante como completa e funcionando uma funcionalidade com bugs conhecidos. "Sim, está pronto... mas tem alguns pequenos bugs" (como Levison [2009] cita ocorrer) não é uma sentença aceitável.

4. Conclusão

Com base na análise dos diversos projetos no instituto, destaca-se a importância de se determinar um processo para o tratamento de bugs em projetos com Scrum, bem como a implementação de boas práticas seguindo a abordagem do comprometimento de toda a equipe. Embora a proposta aqui apresentada ter sido estruturada sobre a metodologia Scrum, nossas conclusões e sugestões podem ser aplicadas a diferentes métodos ágeis.

Outra questão importante a se notar é que é necessário dar a oportunidade da equipe a cometer erros e discutir problemas e possíveis soluções. O comprometimento e a responsabilidade sobre os erros encontrados devem ser discutidos imediatamente por todo o time, e ninguém deve ser responsabilizado responsabilizado (em especial, não só os testadores). Em vez disso, toda a equipe deve analisar o que aconteceu e começar a tomar medidas para evitar a reincidência. Parafraseando Crispin [2009]: se não há tempo suficiente para testar uma nova funcionalidade, então não há tempo para desenvolvê-la.

Conforme já estabelecido na literatura, corrigir um *bug* em uma funcionalidade depois dela estar terminada é sempre mais caro do que fazer certo na primeira vez, então a abordagem de se corrigir dentro do próprio Sprint deve ser a alternativa mais adequada, como em métodos ágeis as entregas são feitas em um curto espaço de tempo os testes exploratórios feitos logo que a versão é disponibilizada permite o conhecimento dos bugs e a sua correção imediata. Outro ponto crucial é a priorização dos bugs em conjunto com o PO e o time, pois assim se leva em consideração a criticidade técnica dos bugs e o valor de negócio para o cliente.

Apesar dos bons resultados obtidos com o modelo proposto, ele ainda pode ser melhorado para permitir que a equipe atinja um melhor desempenho, por exemplo: uma melhor utilização da integração contínua onde os testes unitários seriam gerenciados automaticamente pela ferramenta e análise de métricas para tentar e identificar outras deficiências no processo.

AGRADECIMENTOS

Os autores gostariam de agradecer ao Instituo Nokia de Tecnologia - INdT pela oportunidade em compartilhar estes resultados e todos os membros da equipe em cada projeto por seu apoio no levantamento dos dados.

5. Referências

Bach, J. (2003) Sobre a Empresa: Satisface, INC. Site da Satisface, INC. [Online] http://www.satisfice.com/articles/et-article.pdf.

Beck, K. et al. (2001), "Manifesto for agile software development," [Online]. Available: http://agilemanifesto.org

Cohn, M. (2010) Should Story Points Be Assigned to A Bug-Fixing Story. http://blog.mountaingoatsoftware.com/tag/defects-bugs

Crispin, L.; Gregory, J. (2009) Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley.

Crispin, L. (2003) XP Testing Without XP: Taking Advantage of Agile Testing Practices, 2003. http://www.methodsandtools.com/archive/archive.php?id=2

Feathers, M. (2004) Working Effectively with Legacy Code. Prentice Hall.

Fowler, M. (2001) "The New Methodology," [Online]. Available: http://www.martinfowler.com/articles/newMethodology.html

Glass, R. (2002) Facts and Fallacies of Software Engineering. Addison-Wesley.

Gritti, M. (2010) Regra 10 de Myers [online] http://www.informatiques.blog.br/?p=979

IEEE, (2004) "Guide to the Software Engineering Body of Knowledge", IEEE, Technical Report.

Levison, M. (2009) Coping with Bugs on an Agile/Scrum Project. http://www.infoq.com/agile_techniques

Meszaros, G. (2007) xUnit Test Patterns: Refactoring test code. Addison-Wesley.

Myers, Glenford. (1979) 'The Art of Software Testing.

Opelt, K. (2008) ; Beeson, T. Agile Teams Require Agile QA: How to make it work, an experience report. Agile 2008 Conference

Santamaria, M. (2007) Sobre o site de recursos on-line StickyMinds.com.. [Online] http://www.stickyminds.com/getfile.asp

Santos, A. M. ; Karlsson, B. F. ; Cavalcante, A. M. ; Correia, I. B. ; Silva, E. (2011) Testing in an Agile Product Development Environment: An Industry Experience Report. In proceedings of 12th IEEE Latin-American Test Workshop (LATW), Brazil.

Sommerville, I. (2004) Software Engineering. Pearson/Addison-Wesley.

Sutherland, J. (2009) Practical Roadmap to Great Scrum: Systematically Achieving Hyperproductivity.