Requirements management improvements for an agile project: an experience report

Amanda Sousa[1], Paulo Duarte[1][0000−0001−7604−581X], Ismayle S. Santos[1][0000−0001−5580−643X], Marina Reis Fernandes[2], Mariana Salamoni Francisco[2], and Rossana M. C. Andrade[1][0000−0002−0186−2994]

[1] Group of Computer Network, Software Engineering and Systems - Federal University of Ceará - Fortaleza, Brazil {amandasousa, pauloduarte, ismaylesantos}@great.ufc.br rossana@ufc.br
[2] Furukawa Electric LatAm
{marina.fernandes, mariana.salamoni}@furukawaelectric.com

Abstract. Software requirements management is a process that involves several activities that generate artifacts that serve as a basis for project development. This process occurs continuously and is responsible for performing analysis, tracking, documentation, prioritization and change control so that all stakeholders agree on the requirements. However, these activities are not trivial and their form of execution varies according to the nature of each project. With this in mind, teams should be able to identify points for improvement, such as applying new practices, new technologies or tools and allocating resources to make the process more optimized and more productive. The present work presents our initial results towards improving a requirements engineering process within an agile team. Hence, this paper describes an experience report about identifying and applying good practices within a software development project. The difficulties and main lessons learned are also discussed.

Keywords: Requirements · Management · Traceability.

1 INTRODUÇÃO

Em tempos recentes, métodos ágeis, como o Scrum [10] e o Extreme Programming (XP) [2], tornaram-se populares no desenvolvimento de software. Dentre os motivos, pode-se mencionar o *feedback* contínuo, a entrega incremental e a adaptação às mudanças nos requisitos [7].

Dependendo da complexidade do projeto, os requisitos podem ser especificados com casos de uso (UC, do inglês *use case*), protótipos e diagramas UML. A vantagem dos casos de uso reside na sua flexibilidade e no fato deles permitirem documentar mais precisamente os requisitos [6]. Nesse sentido, Heikkila *et al.* [3] indicam que, quando sistemas de software complexos são desenvolvidos com métodos ágeis, as práticas tradicionais de engenharia de requisitos como "gerenciamento da rastreabilidade dos requisitos" devem ser seguidas. Além disso, os autores apresentam outros desafios dos requisitos no desenvolvimento de software ágil, tais como priorização dos requisitos e o crescimento de dívidas técnicas.

Logo, mesmo para métodos ágeis, devem ser utilizadas práticas do gerenciamento de requisitos, tais como [11]: rastrear versões de requisitos, gerenciar mudanças, acompanhar o status dos requisitos e manter a rastreabilidade dos requisitos com outros artefatos. Nesse cenário, diferentes abordagens foram propostas para os desafios envolvendo requisitos no ambiente ágil, tais como rastreabilidade entre testes e requisitos[9], rastreabilidade entre requisitos não funcionais [1] e mudança de requisitos [4].

O intuito deste artigo é descrever um relato de experiência sobre a melhoria do processo de gerenciamento de requisitos no contexto de um projeto de desenvolvimento de software usando Scrum. Durante o período observado, foram mapeados os desafios enfrentados pela equipe de requisitos do projeto, e foram avaliadas estratégias de mitigação para os problemas identificados. Os desafios e as soluções aplicadas são apresentadas e discutidas neste artigo.

O restante deste trabalho vai ser organizado da seguinte forma: na Seção 2, são apresentados trabalhos relacionados. A Seção 3 contextualiza o projeto e a equipe. As Seções 4 e 5 relatam os problemas identificados após as primeiras *Sprints* e as lições aprendidas com essas ações. Por fim, a Seção 6 apresenta as considerações finais.

2 TRABALHOS RELACIONADOS

Panis sumarizou, de forma empírica, os problemas encontrados na etapa de desenvolvimento de um novo projeto que podiam ser rastreados a falhas em um processo de engenharia de requisitos que se acreditava maduro e estabelecido. Além disso, o autor apresentou as soluções que implementaram para mitigar tais dificuldades no seu processo. Diferente de Panis, o presente artigo relata a sumarização de problemas encontrados empiricamente após a realização de 9 *Sprints* em um projeto de desenvolvimento de software na indústria [8].

Mezghani *et al* [5] propuseram uma abordagem utilizando a técnica k-means para automatizar a geração de uma matriz de rastreabilidade. Enquanto os autores priorizaram o enfoque na rastreabilidade dos requisitos, o presente artigo prioriza modificações no processo de requisitos e na comunicação entre as equipes, identificando problemas (ou os recebendo via *feedback* das equipes) e validando soluções adotadas.

3 AMBIENTE DA EXPERIÊNCIA

3.1 O Projeto

O projeto se concentra no desenvolvimento de um sistema de gerenciamento de falhas em redes GPON *(Gigabit Passive Optical Network)*. Esse sistema tem o objetivo de processar informações de baixo nível, que são recebidas em tempo real pelos equipamentos que o usuário deseja monitorar, e entregar para o usuário final dados legíveis e significativos que melhorem as tomadas de decisão em relação

a resolução de problemas em campo. O sistema em desenvolvimento possui integrações com outros módulos desenvolvidos em outros projetos da mesma empresa. Nesse contexto, existe um certo nível de complexidade quanto aos seus requisitos.

A equipe do projeto é composta por um analista de requisitos, três testadores, cinco desenvolvedores, um PO (*Product Owner*), um gerente e, houve por um período, dois consultores que eram especialistas no domínio do sistema em desenvolvimento. O time opera de forma distribuída e remota. Em uma ponta, o projeto possui o time de desenvolvimento que é um grupo separado, que possui seu próprio local de trabalho e atua sob supervisão de um líder técnico.

Em outra ponta, o projeto conta com os times de desenvolvimento e testes que pertencem a um mesmo grupo e respondem a autoridade de um gerente de projetos. Por fim, como cliente, o projeto possui um P.O da empresa contratante que também dispõe de outros profissionais que são suporte a processos de integração de módulos do sistema com outros sistemas da empresa, design, marketing e vendas.

O processo de comunicação da equipe funciona como um fluxo contínuo, com o apoio de ferramentas como Slack, Skype, Google Meets e, para conversas mais informais e internas das equipes de requisitos e testes, os times também fazem uso do aplicativo Whatsapp.

O projeto de desenvolvimento de software relatado neste trabalho possui um processo híbrido, contendo aspectos ágeis fundamentados no SCRUM e aspectos tradicionais e é composto em, uma visão macro, por atividades de engenharia de requisitos, testes e desenvolvimento. O processo é iterativo, dividido em *sprints* de 4 semanas e o entregável é apresentado para os *stakeholders*. Durante as sprints, os requisitos são elicitados, analisados, documentados e atualizados, e dessa forma, consequentemente os backlogs do produto e da *sprint* são atualizados conforme a prioridade de cada tarefa.

3.2 Processo Utilizado

O processo utilizado possui 4 fases principais, a saber: Elicitação de requisitos, Especificação, Validação e Gerenciamento de Requisitos, conforme ilustra a Figura 1.

A atividade de **Elicitação de requisitos** trata de identificar novos requisitos e discuti-los com o PO do projeto a fim de incluí-los ou descartá-los do escopo das atividades. Essa atividade é executada na reunião de *kickoff* que é realizada no início do projeto e durante as sprints, possuindo caráter semanal. A partir dessa atividade é possível manter as expectativas do cliente atualizadas e documentadas. Os artefatos de entrada dessa fase são constituídos pelas informações obtidas com o PO do projeto por meio de entrevistas semi estruturadas. O artefato produzido nessa fase é um conjunto de um ou mais requisitos brevemente descritos e priorizados que serão insumos para fase de especificação.

Na fase de **Especificação** são realizadas atividades de elaboração de protótipos de baixa fidelidade, bem como a especificação de requisitos, casos de uso, regras negócios e demais elementos da documentação necessários. Todos os in-

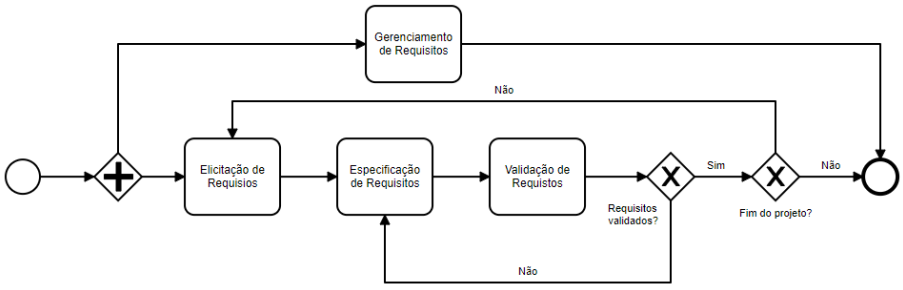

Fig. 1. Processo de Requisitos

sumos dessa atividades são **validados** durante as reuniões semanais de acompanhamento das atividades com todos os times (Requisitos, Desenvolvimento e Testes) do projeto na presença dos gerentes e PO.

A especificação e documentação dos requisitos é feita em uma Wiki da ferramenta Confluence[3], permitindo acesso a todos os membros do projeto. Nessa Wiki, são documentados requisitos funcionais, não funcionais, regras de negócio, casos de uso e demais informações necessárias sobre o sistema. Esses requisitos são convertidos em tasks de implementação no sistema de gerenciamento das atividades da sprint (software Jira[4]) e, além disso, geram tasks de planejamento e execução de testes.

Gerenciamento dos requisitos ocorre de forma contínua durante as sprints do projeto, tratando das atividades de análise de impacto de mudanças dos requisitos, controle e rastreamento de alterações e repriorização de tarefas e requisitos. Na análise de impactos, em decorrência de mudanças de requisitos, são avaliados os módulos que necessitarão de alterações e que atividades deverão ser realizadas por cada time do projeto. Em seguida, essas informações são levadas para as reuniões semanais para que sejam avaliadas pelo P.O e pelos times, e são, por fim, priorizadas. Antes, o controle e rastreamento de alterações era feito apenas utilizando os históricos das páginas da Wiki.

4 MUDANÇAS NO PROCESSO DE ER

Com a utilização do processo descrito, utilizado durante as 09 primeiras *Sprints*, foram encontrados alguns problemas.

– **(P1) Falta de referências para alterações em requisitos existentes.** Alterações nos artefatos de requisitos podem ter baixo ou alto impacto. Um exemplo de baixo impacto são pequenas correções, enquanto um exemplo de alto são mudanças em fluxos importantes de requisitos ou casos de uso (alto

[3] https://www.atlassian.com/br/software/confluence
[4] https://www.atlassian.com/br/software/jira

impacto). Como a documentação é armazenada em uma Wiki que disponibiliza o histórico da página, os membros do projeto não se preocupavam em explicitar alterações. Bastava apenas verificar o histórico de mudanças pela própria ferramenta da Wiki. No entanto, foi observado que esse processo revelou-se dispendioso e pouco prático para testadores e desenvolvedores. Dessa forma, por não corresponder a tais mudanças, o código do sistema, os requisitos e os casos de testes apresentavam inconsistências entre si sempre que o caso de uso precisava ser atualizado. Em decorrência disso, o sistema tinha dificuldades em atingir os requisitos de forma assertiva, enquanto os testes inconsistentes geravam diversos *bugs* inválidos. Isso, por sua vez, gerava retrabalho e frustração do time.

- **(P2) Falta de rastreabilidade entre UCs, CTs e código.** Na documentação dos requisitos do projeto, não existia uma matriz de dependência entre os requisitos. Dessa forma, havia a dificuldade de controlar possíveis alterações que devam ocorrer em decorrência de outras mudanças (efeito cascata). A falta de controle entre as dependências dos requisitos gera inconsistências na documentação. Isso leva à dificuldades em implementar o sistema e elaborar os planos de testes, uma vez que os requisitos passam a ser conflitantes entre si.
- **(P3) Atualização de requisitos.** A manutenção e a consistência da rastreabilidade bidirecional entre casos de uso e casos de testes traziam desafios à equipe de requisitos. No processo utilizado, por vezes os casos de testes ficavam desatualizados, devido à dificuldade de verificar a atualização e implementação dos requisitos do sistema. Por exemplo, ao atualizar um UC, esse era validado e liberado para implementação e os casos de testes eram atualizados. Entretanto, quando surgiam alterações nesses casos de uso (por exemplo, a diminuição do seu escopo na implementação por meio de acordos com o PO no meio da *Sprint*), isso não era visto pelo time de testes, ocasionando a abertura de *bugs* inválidos ou até diminuição da cobertura do teste.

4.1 Práticas

Com o objetivo de mitigar os problemas identificados foram propostas soluções no formato de aplicação de práticas acordadas com a equipe.

(S1) Documentação da dependência dos requisitos por meio de links da Wiki. Foi adotada a prática de utilizar links para todos os itens relacionados a cada requisito. Os requisitos são documentados de forma macro e de forma específica e hierárquica. Cada requisito possui links para todos os casos de uso associados. Dentro dos casos de uso são identificadas todas as telas dos protótipos de baixa fidelidade (*mockups*). Os *mockups* são a na maioria das vezes a fonte primária de todas as alterações, dessa forma, com essa estrutura é possível visualizar quais casos de uso e requisitos precisarão ser alterados caso haja mudanças nos *mockups*. Caso a alteração seja em um determinado caso de uso do sistema,

também é possível, dentro do próprio caso de uso, identificar que outros casos de uso precisam ser alterados, pois os links para os casos de uso relacionados e regras de negócio também encontram-se em seções no próprio caso de uso. Além disso, no projeto, fez-se necessário documentar as mensagens do sistema para dar suporte em diferentes linguagens. Esses itens são documentados indicando o local das mensagens e o texto correspondente para cada linguagem disponível no sistema. Dessa forma ao realizar uma alteração nas telas do sistema que tenha impacto nas mensagens ou em algum texto do sistema com o suporte *multi-language* é fácil identificar o local de alteração na documentação. Essa prática endereça o problema de rastreabilidade entre os requisitos (P3).

(S2) Arquivo de *changelog* de cada *Sprint*. O *changelog* foi adotado como forma de manter o histórico de alterações dos requisitos e também ser uma fonte para que todos os times tenham acesso e sejam capazes de agir de forma necessária para acompanhar os requisitos. As alterações são documentadas por data e são descritas contendo o *link* do item alterado e uma breve descrição da alteração realizada. Cada alteração é sinalizada com uma *tag* que indica o tipo de mudança efetuada no item.

Em cada *Sprint* é criada uma página para documentar o *changelog* dos requisitos, dessa forma mantém-se a rastreabilidade das alterações em cada *Sprint* e também permite que desenvolvedores e testadores acompanhem as informações de forma rápida. A página de *changelog* permite que os participantes habilitem notificações caso ela seja alterada e dessa forma toda a equipe pode imediatamente ficar ciente das mudanças efetuadas na documentação.

5 LIÇÕES APRENDIDAS

L1: *Board* de Requisitos separado visando melhor acompanhamento das atividades.

Evento: As atividades de requisitos e desenvolvimento ficavam no mesmo *board*, que era adaptado ao fluxo dos desenvolvedores. As pontuações de requisitos e desenvolvimento requeriam cálculos extras que atrapalhavam o planejamento. **Ação**: Foi criado um *backlog* próprio, seguindo o fluxo das atividades de requisitos. O fluxo utiliza a nomenclatura própria ("Em Análise", "Opções", "Em Especificação", "Pronto para Revisão", "Em Revisão", "Pronto para Validação", "Validado", "Concluído", e "Descartado") de forma a permitir o planejamento de pequeno, médio e longo prazo da equipe de requisitos sem interferir ou misturar com a pontuação das equipes de desenvolvimento e testes.

L2: Necessidade de manter a rastreabilidade entre requisitos e artefatos.

Evento: Foram identificados problemas de rastreabilidade entre as versões mais atuais dos requisitos (cuja implementação deveriam ocorrer no médio prazo) e as versões a serem implementadas pela equipe de desenvolvimento na *Sprint* atual. **Ação**: A versão a ser implementada passou a inserida diretamente na *issue* do Jira. Além disso, planeja-se o desenvolvimento da matriz de rastreabilidade

dos requisitos e a utilização da ferramenta U-Til[9] para auxiliar a equipe de testes a encontrar os casos de testes impactados por possíveis novos requisitos.

6 CONSIDERAÇÕES FINAIS

Este artigo apresentou um relato de experiência sobre práticas aplicadas para melhorar o processo de gerenciamento de requisitos no contexto de um projeto com a indústria que utilizada o Scrum. Além de descrever os desafios e práticas testadas durante o período de observação, foram discutidos no artigo algumas lições aprendidas. Como trabalhos futuros, pretende-se automatizar a matriz de rastreabilidade entre os requisitos, testes e código-fonte, de modo a facilitar atividades relacionadas aos testes de regressão e refatoração.

References

1. Arbain, A.F., Jawawi, D.N.A., Ghani, I., Kadir, W.M.W.: Non-functional requirement traceability process model for agile software development. Journal of Telecommunication, Electronic and Computer Engineering (JTEC) **9**(3-5), 203–211 (2017)
2. Beck, K.: Extreme programming explained: embrace change. addison-wesley professional (2000)
3. Heikkilä, V.T., Damian, D., Lassenius, C., Paasivaara, M.: A mapping study on requirements engineering in agile software development. In: 2015 41st Euromicro conference on software engineering and advanced applications. pp. 199–207. IEEE (2015)
4. Madampe, K., Hoda, R., Grundy, J.: A faceted taxonomy of requirements changes in agile contexts. IEEE Transactions on Software Engineering (2021)
5. Mezghani, M., Kang, J., Kang, E.B., Sedes, F.: Clustering for traceability managing in system specifications. In: 2019 IEEE 27th International Requirements Engineering Conference (RE). pp. 257–264 (2019). https://doi.org/10.1109/RE.2019.00035
6. Nawrocki, J., Ochodek, M., Jurkiewicz, J., Kopczyńska, S., Alchimowicz, B.: Agile requirements engineering: A research perspective. In: International Conference on Current Trends in Theory and Practice of Informatics. pp. 40–51. Springer (2014)
7. Ozkan, N., Gök, M.Ş., Köse, B.Ö.: Towards a better understanding of agile mindset by using principles of agile methods. In: 2020 15th Conference on Computer Science and Information Systems (FedCSIS). pp. 721–730. IEEE (2020)
8. Panis, M.C.: An analysis of requirements-related problems that occurred in an organization using a mature requirements engineering process. In: 2020 IEEE 28th International Requirements Engineering Conference (RE). pp. 291–299. IEEE (2020)
9. Santos, I.S., Lelli, V., Duarte, P., Nogueira, T.P., Araújo, J.P., Andrade, R.M.C.: U-til: lightweight approach for traceability management between user stories and test suites. In: 2021 16th Iberian Conference on Information Systems and Technologies (CISTI). pp. 1–7 (2021). https://doi.org/10.23919/CISTI52073.2021.9476601
10. Sutherland, J., Schwaber, K.: The scrum guide. The definitive guide to scrum: The rules of the game. Scrum. org **268** (2013)
11. Wiegers, K., Beatty, J.: Software requirements. Pearson Education (2013)