Aluno: Bernardo de Resende Marcelino
Disciplina: Projeto de software
Professor: João Paulo Aramuni
Resenha Crítica "Documenting Architecture Decisions"

O artigo "Documenting Architecture Decisions" de Michael Nygard busca estabelecer uma abordagem prática e sustentável para documentação de decisões arquiteturais em projetos de software ágil, reconhecendo que métodos ágeis não se opõem à documentação em si mas sim à documentação sem valor, propondo que decisões arquiteturalmente significativas sejam capturadas através de registros modulares, concisos e evolutivos chamados Architecture Decision Records que preservem o contexto, a motivação e as consequências de escolhas arquiteturais importantes ao longo do ciclo de vida do projeto, permitindo que desenvolvedores compreendam o raciocínio por trás de decisões passadas e avaliem adequadamente quando e como essas decisões devem ser revisitadas ou modificadas conforme o contexto do projeto evolui.

Nygard identifica um problema fundamental no desenvolvimento de software onde uma das coisas mais difíceis de rastrear durante a vida de um projeto é a motivação por trás de certas decisões arquiteturais, observando que quando uma nova pessoa ingressa em um projeto ela pode ficar perplexa, confusa, encantada ou irritada com alguma decisão passada, mas sem compreender a justificativa ou as consequências dessa decisão, essa pessoa tem apenas duas escolhas igualmente problemáticas, sendo a primeira aceitar cegamente a decisão, resposta que pode ser adequada se a decisão ainda é válida mas que se torna prejudicial quando o contexto mudou e a decisão deveria realmente ser revisitada, levando ao acúmulo de decisões aceitas sem compreensão que eventualmente torna a equipe de desenvolvimento temerosa de mudar qualquer coisa fazendo o projeto colapsar sob seu próprio peso, ou alternativamente mudar cegamente a decisão, abordagem que novamente pode ser adequada se a decisão precisa ser revertida mas que ao mudar a decisão sem entender sua motivação ou consequências pode significar danificar o valor geral do projeto sem perceber, como quando a decisão suportava um requisito não funcional que ainda não havia sido testado, estabelecendo assim a necessidade de evitar tanto aceitação cega quanto reversão cega através de documentação apropriada que capture e preserve o conhecimento arquitetural.

Diante desse cenário, o autor propõe manter uma coleção de registros para decisões "arquiteturalmente significativas", ou seja, aquelas que afetam a estrutura, características não funcionais, dependências, interfaces ou técnicas de construção do sistema, definindo que um Architecture Decision Record é um arquivo de texto curto em formato similar a um padrão Alexandrino onde embora as decisões em si não sejam necessariamente padrões, elas compartilham a característica de balancear forças, descrevendo cada registro um conjunto de forças e uma única decisão em resposta a essas forças, estabelecendo que a decisão é a peça central de forma que forças específicas podem aparecer em múltiplos ADRs, propondo que os ADRs sejam mantidos no repositório do projeto sob diretório específico, utilizando linguagem de formatação de texto leve como Markdown ou Textile, numerados sequencialmente e monotonicamente onde números não são reutilizados, preservando decisões antigas mesmo quando revertidas marcando-as como substituídas pois ainda é relevante saber que aquela foi a decisão mesmo que não seja mais, utilizando

formato com apenas algumas partes para que cada documento seja fácil de digerir, incluindo título como frase nominal curta, contexto descrevendo as forças em jogo incluindo aspectos tecnológicos, políticos, sociais e locais do projeto em linguagem neutra simplesmente descrevendo fatos, decisão descrevendo a resposta a essas forças em sentenças completas com voz ativa, status indicando se a decisão está proposta, aceita, depreciada ou substituída com referência ao seu substituto, e consequências descrevendo o contexto resultante após aplicar a decisão listando todas as consequências não apenas as positivas reconhecendo que uma decisão particular pode ter consequências positivas, negativas e neutras mas todas afetam a equipe e o projeto no futuro.

O autor enfatiza que o documento completo deve ter uma ou duas páginas sendo escrito como se fosse uma conversa com um desenvolvedor futuro, requerendo bom estilo de escrita com sentenças completas organizadas em parágrafos, aceitando bullets apenas para estilo visual e não como desculpa para escrever fragmentos de sentença, estabelecendo que um ADR descreve uma decisão significativa para um projeto específico sendo algo que tem efeito sobre como o resto do projeto será executado, reconhecendo que as consequências de um ADR são muito prováveis de se tornarem o contexto para ADRs subsequentes de forma similar à ideia de Alexander de linguagem de padrões onde respostas de larga escala criam espaços para a escala menor se encaixar, permitindo que desenvolvedores e stakeholders do projeto vejam os ADRs mesmo quando a composição da equipe muda ao longo do tempo, tornando a motivação por trás de decisões anteriores visível para todos presentes e futuros de modo que ninguém fica confuso tentando entender o que estavam pensando e o momento de mudar decisões antigas fica claro a partir de mudanças no contexto do projeto.

Nygard relata experiência prática de uso do formato ADR em diversos projetos desde agosto anterior à publicação do artigo, observando que embora não seja um tempo muito longo no sentido global, o feedback inicial tanto de clientes quanto de desenvolvedores tem sido bastante positivo, destacando que nesse período seis a dez desenvolvedores rotacionaram através de projetos usando ADRs e todos declararam que apreciam o grau de contexto que receberam ao lê-los, notando que ADRs têm sido especialmente úteis para capturar intenções de longo prazo em clientes que estão estabilizando seus sistemas atuais mas olhando para uma “rearquitetura” maior em futuro não muito distante, permitindo que ao escrever essas intenções a equipe não torne inadvertidamente essas mudanças futuras mais difíceis, abordando potencial objeção de que manter esses registros em controle de versão com o código os torna menos acessíveis para gerentes de projeto, stakeholders de clientes e outros que não vivem em controle de versão como a equipe de desenvolvimento, argumentando que na prática como projetos geralmente vivem em repositórios privados do GitHub é possível trocar links para a versão mais recente e como GitHub processa markdown automaticamente o resultado parece tão amigável quanto qualquer página de wiki.

Dessa forma, o artigo de Nygard estabelece uma contribuição prática e imediatamente aplicável para a documentação arquitetural em projetos de software ao propor um formato leve, modular e evolutivo que equilibra a necessidade de preservar conhecimento arquitetural importante com os princípios ágeis de documentação apenas quando agrega valor, demonstrando que documentação eficaz não precisa ser extensa ou complexa mas sim focada em capturar decisões significativas junto com seu contexto e consequências de

forma que permaneça relevante e acessível ao longo da evolução do projeto, proporcionando mecanismo que previne tanto a ossificação de decisões arquiteturais através de aceitação cega quanto a erosão arquitetural através de mudanças desinformadas, criando memória institucional que sobrevive à rotatividade de equipes e evolução de contexto, representando assim abordagem que reconhece que arquitetura em projetos ágeis não é conjunto de decisões feitas todas de uma vez no início mas sim processo evolutivo onde decisões são tomadas continuamente e precisam ser documentadas.