Artigo 9 - Documenting Architecture Decisions

Nome do estudante: João Victor Filardi Souza Pinto

O artigo *Documenting Architecture Decisions*, de Michael Nygard, apresenta uma abordagem prática e leve para registrar decisões arquiteturais em projetos de software, especialmente em contextos ágeis. O autor parte do princípio de que a arquitetura, em projetos desse tipo, não é definida de uma vez por todas no início do desenvolvimento, mas evolui ao longo do tempo conforme novas informações, restrições e aprendizados surgem. Por isso, a forma de documentar precisa acompanhar essa natureza dinâmica, priorizando clareza, concisão e valor real para a equipe.

Nygard critica os documentos extensos e complexos que tradicionalmente tentavam capturar toda a arquitetura do sistema. Segundo ele, esses materiais acabam se tornando obsoletos rapidamente, pois raramente são atualizados, além de serem pouco lidos e compreendidos pelos desenvolvedores. Em vez disso, ele propõe o uso de **Architecture Decision Records (ADRs)** — registros curtos e modulares que documentam uma decisão arquitetural relevante e o contexto que levou a ela. Cada ADR deve caber em uma ou duas páginas e ter como público principal os próprios desenvolvedores e futuros membros da equipe.

O formato sugerido é simples e segue uma estrutura inspirada nos padrões de Christopher Alexander. Cada ADR contém cinco seções principais: **Título**, **Contexto**, **Decisão**, **Status** e **Consequências**. O *Contexto* descreve as forças e restrições que influenciam a decisão, como limitações tecnológicas, exigências de negócio ou prazos. A *Decisão* apresenta de forma direta o caminho escolhido (“Nós iremos…”), enquanto o *Status* indica se ela foi proposta, aceita, ou substituída. Já as *Consequências* listam os impactos resultantes, sejam positivos ou negativos, de modo a deixar claras as implicações futuras da escolha.

Um ponto fundamental defendido pelo autor é que cada ADR deve registrar **apenas uma decisão significativa**, de modo que seja fácil localizar, revisar ou substituir esse registro no futuro. Quando uma decisão for revista, o ADR anterior não é apagado, mas marcado como “superado”. Isso preserva o histórico de pensamento do time e ajuda novos desenvolvedores a compreenderem as razões por trás de decisões antigas — evitando que escolhas sejam aceitas ou revertidas “às cegas”. Essa transparência ajuda a manter o conhecimento arquitetural vivo e acessível, mesmo com trocas de pessoal ao longo do tempo.

O autor também compartilha uma breve experiência prática: em projetos que aplicaram o formato de ADRs desde 2011, tanto desenvolvedores quanto clientes demonstraram satisfação com o resultado. As equipes perceberam que os registros ajudavam a capturar intenções de longo prazo, facilitando o entendimento e a evolução da arquitetura. Além disso, armazenar esses documentos junto ao código, em repositórios como o GitHub, mostrou-se eficiente, já que o histórico fica versionado e facilmente acessível a todos os envolvidos.

Na prática, os princípios apresentados por Nygard podem ser aplicados com grande valor em empresas que adotam metodologias ágeis e buscam equilíbrio entre flexibilidade e rastreabilidade técnica. Por exemplo, em uma equipe de desenvolvimento de um sistema de e-commerce, decisões como “adotar microsserviços”, “utilizar mensageria assíncrona” ou “armazenar logs em nuvem” podem ser registradas em ADRs. Dessa forma, quando novos membros ingressam no projeto, eles compreendem rapidamente as motivações de cada escolha, reduzindo o tempo de adaptação e evitando retrabalho. Além disso, ao longo da evolução do sistema, revisar um ADR torna-se um processo simples e controlado, preservando o raciocínio histórico que levou à decisão inicial.

Em resumo, o artigo de Nygard mostra que documentar arquitetura não precisa ser sinônimo de burocracia. Pelo contrário, ele propõe um formato leve, útil e duradouro, que privilegia o entendimento e a comunicação entre os membros da equipe. Ao transformar cada decisão importante em um pequeno registro vivo, os ADRs se tornam uma ferramenta essencial para garantir coerência, aprendizado e evolução contínua nos projetos de software.