1 A Case Study of API Redesign for Improved Usability (Jeffrey Stylos et al.)

[1] Stylos, B. Graf, D. K. Busse, C. Ziegler, R. Ehret, and J. Karstens, "A case study of API redesign for improved usability," in 2008 IEEE Symposium on Visual Languages and Human-Centric Computing, Sep. 2008, pp. 189–192, doi: 10.1109/VLHCC.2008.4639083.

Descrição Geral: Apresenta um estudo de caso de redesign de uma API. O foco é um processo de design centrado no usuário. Ou seja, colocar os requisitos e necessidades do usuário em foco durante o processo de desenvolvimento. Acho que é um trabalho que mais se aproxima do nosso, por se tratar de um estudo de caso e de um método formal com etapas a serem seguidas.

Pontos Positivos: Estudo misto de dados empíricos e analíticos. Na etapa de requisitos eles usam o conceito de personas, mas a criação dos objetivos dos usuários vem muito mais das entrevistas realizadas com potenciais usuários do que do suporte efetivo do método. na etapa seguinte eles propõe a criação de "pseudocódigos" para realizar testes de usabilidade com usuários reais. E finaliza com uma avaliação de usabilidade da interface. É um estudo rico em dados empíricos.

Pontos negativos: O artigo não formaliza um método propriamente. Ficou mais parecendo um conjunto de "lições aprendidas" do estudo de caso. Aparentemente o método emergiu do estudo de caso e não foi testado neste estudo de caso.

Diferenças com nosso trabalho: O trabalho é uma proposta voltada para estudos com a presença de usuários, coleta de dados empíricos, entrevistas e etc. Ou seja, todo método depende exclusivamente de um massivo trabalho com participantes para ser executado. O nosso método, por outro lado, foca no poder epistêmico das ferramentas utilizadas, ou seja, o método oferece apoio para que o designer possa refletir, na primeira etapa sobre os requisitos, e na segunda etapa sobre os fluxos de conversas.

2 Improving Software API Usability through Text Analysis: A Case Study (Robert B. Watson)

[2] R. B. Watson, "Improving software API usability through text analysis: A case study," in 2009 IEEE International Professional Communication Conference, Jul. 2009, pp. 1–7, doi: 10.1109/IPCC.2009.5208679.

Descrição Geral: Apresenta a análise de texto técnica como uma alternativa a melhorar a usabilidade de APIs. Ele apresenta um método para analisar as interfaces da API de modo a identificar inconsistências que poderiam prej-

udicar a usabilidade. Neste método, ele decompõem os elementos e analisa se o nome está compatível com tipo de dado, analisa o nome dos métodos está compatível com o que o método faz e por fim analisa os métodos compostos de get e set.

Pontos positivos: É um método formal e organizado para identificar inconsistências no código fonte da API. Utilizando a análise técnica, ele propões algumas diretrizes para isso. Por exemplo, métodos que retornam valores booleanos devem estar em "voz ativa" para uma melhor legibilidade do código fonte. Através da aplicação direta do método, certamente podemos ter melhores interfaces e ganhos com a usabilidade.

Ponto negativo: Eu achei superficial. Trata apenas um ponto da questão - nomenclatura. Usabilidade em uma API vai muito além disso, e o método deixa esse ponto de fora. Por exemplo, usabilidade está relacionado também ao fluxo de chamadas em uma API, o que fica totalmente de fora da análise.

Diferença com nosso trabalho: Há duas diferenças principais. A primeira o fato dele deixar de fora o fluxo de interação do usuário com a API, algo que tratamos no nosso trabalho através da modelagem com a MoLIC. Outro ponto que ficou de fora é a questão pragmática do significado dos nomes. Ele trás diretrizes interessantes para a nomenclatura de métodos, porém isso não implica necessariamente em melhor entendimento do raciocínio do designer. Por exemplo, não usar o verbos "is" em uma função booleana como ele sugeriu (não usar IsValid(), e sim Valid()) pode melhorar a legibilidade do código, mas não muda em nada o entendimento do que significa, naquele contexto, "ser válido". Nosso método, ao utilizar técnicas com a MoLIC e o template de metacomunicação (perguntas guia), vai além, e faz o designer da API refletir sobre o significado inserido em suas interfaces e as possíveis interpretação dos diferentes tipos de usuários.

3 API: Design Matters (MICHI HENNING)

[**3**] M. Henning, "API Design Matters," Queue, vol. 5, no. 4, pp. 24–36, May 2007, doi: 10.1145/1255421.1255422.

Descrição Geral: Este trabalho apresenta um conjunto de diretrizes para promover a usabilidade de uma API durante a fase de design. Além disso, o autor faz uma extensa discussão do impacto e da necessidade de um bom design para APIs. Seus impactos nos usuários e nos custos com problemas nos softwares gerados.

Pontos positivos: Oferece diretrizes que vão além de apenas nomear elementos da API. Suas diretrizes apontam para um processo de design que vai desde a documentação até a implementação da API.

Pontos negativos: Algumas diretrizes são bastante genéricas. Por exemplo "An API must provide sufficient functionality". O que é suficiente? Assim, algumas diretrizes são fracas e podem não auxiliar propriamente o designer.

Diferença para o nosso trabalho: Vamos além do proposto pelo autor ao definir um método que vai além de um conjunto de diretrizes. Por exemplo, enquanto o autor oferece a diretriz que diz que a "API precisa prover funcionalidade suficiente", nosso método oferece uma linguagem de modelagem que vai permitir ai designer refletir sobre quais são essas funcionalidades. E isso foi exatamente o que aconteceu com nosso estudo de caso. Temos relato do designer da API dizendo que só percebeu a necessidade de algumas funcionalidades a partir do momento que modelou usando a MoLIC4APIs.

4 How to Design a Good API and Why it Matters (Joshua Bloch)

[**4**] J. Bloch, "How to design a good API and why it matters," in Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, Portland, Oregon, USA, Oct. 2006, pp. 506–507, doi: 10.1145/1176617.1176622.

Descrição Geral: Apresenta um conjunto de diretrizes que emergiram da prática, as quais ele chama de "máximas de design". Neste artigo não é proposto um método de design propriamente, mas um conjunto de regras que, se seguidas, garantem o sucesso do design de APIs.

Ponto positivo: Simples de usar pelos Engenheiros de Software. Aponta para problemas relativamente frequentes no design de APIs, por exemplo, nomenclatura de métodos, numero de parâmetros e etc.

Ponto negativo: Um conjunto apenas de recomendações, que não alinham um método que garantirá sucesso de design.

Diferença com nosso trabalho: As diretrizes citadas não apontam em momento nenhum para os "desentendimentos" entre usuário e design. Dar bons nomes para um método não garante que o entendimento pelos usuários vai ser o mesmo que o designer quis "falar". Nosso trabalho se propõem a ir além disso. Colocar "signos" na interface para chamar atenção do usuário para o raciocínio do designer por trás dos nomes da interface.