Edeyson A. Gomes

Software Aging

Resumo

A intangibilidade do software obscurece o fato de que o mesmo sofre de obsolescência temporal causada pela natureza dinâmica de sua adaptação a novos requisitos, correção, etc. que contribuem para a sua deterioração.

Como para qualquer artefato tangível, a inquietação com o envelhecimento do software desperta a busca do entendimento de suas causas como fundamento para que se determine como limitar seus efeitos, reverter os prováveis danos causados e como prepará-lo para a obsolescência.

Dentre as causas do envelhecimento do software destacam-se a falha em manutenção evolutiva para abarcar novos requisitos e o resultado de mudanças efetuadas. Numa relação de causa e conseqüência, a primeira, se efetuada por desenvolvedores com visão pontual do projeto leva à degradação da estrutura do software, tornando-o desconhecido à equipe original de desenvolvimento. Este processo, se cíclico, minimizará a compreensão de todo o produto, onerando futuras atualizações.

O envelhecimento do software pode acarretar em ônus ocasionado por fatores como a perda de clientes para novos produtos, pois a atualização de software envelhecido é normalmente cara e passível de injeção de falhas. Ademais, a manutenção tende a aumentar o tamanho do código, pois a forma mais fácil de adicionar recurso é com código novo. Como resultado desses fatores, tem-se a degradação do desempenho e o consumo de mais recursos, como memória, contribuindo para o descarte do software.

Constatado que o envelhecimento do software é fato, buscam-se critérios que reduzam seu impacto, com o foco em seu futuro. Numa analogia com a medicina preventiva em humanos, cabe determinar como retardar o envelhecimento do software e limitar seus efeitos.

O foco no futuro do software induz ao projeto voltado para mudanças. Este princípio é negligenciado na indústria, onde o desenvolvedor – possivelmente com formação inadequada quanto à base teórica – é pressionado por prazos para a entrega de produto funcional. Isto o leva a minimizar a preocupação com o custo futuro de mudança, negligenciando documentação e revisão de projeto e código.

Uma documentação acurada, facilmente localizada e manutenível, que registre os princípios e decisões de projeto, facilita a manutenção futura do software e é fator para controle de seu envelhecimento. A assertiva de que o código se documenta é nociva a sua longevidade. É constatável que a elaboração de documentação de projeto com acurácia é raro, pois o foco maior normalmente é dado na documentação de código.

Não obstante uma boa documentação, outro fator de destaque à longevidade do software é a revisão. Comum em diversas áreas do conhecimento, como medicina, construção

civil e construção naval, a revisão de projeto não tem a devida atenção na produção de software. Uma segunda opinião de um especialista não imerso no projeto pode detectar falhas ou oportunidades ainda não exploradas.

Ainda que se advogue a prevenção do envelhecimento, o tratamento é imprescindível quando se aborda software legado, que pode estar deteriorado como efeito de manutenção. Neste caso, objetiva-se retardar a deterioração através de princípios como documentação revisada que valide a consistência do código. A retroação na elaboração da documentação garante o rejuvenescimento do código e, se elaborado via análise sistemática deste, pode-se acrescentar o benefício da descoberta de problemas ocultos, eliminando-os.

Outros aspectos a considerar são a retroação na decomposição modular do código, a remoção de código desnecessário e sua generalização, possibilitando compartilhar mudanças em família de produtos de um mesmo domínio de aplicação.

O reconhecimento de que o envelhecimento do software é realmente um problema deve conduzir a um novo estilo de desenvolvimento, focado no planejamento para mudança. Embora a urgência e o imediatismo na entrega do produto final de software estresse os desenvolvedores, a máxima de pensar no futuro e que o mesmo traz mudanças deve ser seguida. Planejar para mudança consiste em projetar, documentar e revisar antes de codificar. A concomitância entre o projeto e a documentação é fundamental para a acurácia dessa.

Projetar para o futuro também deve visar e planejar a substituição do software, dado que o mercado já convive com a obsolescência programada de produtos eletrônicos, carros, etc., mas esse conceito é pouco usual quando se refere a software. Outra divergência com o mercado é apontada na afirmação de que a indústria de software é diferente das demais, pois produz o isolamento intelectual entre nichos onde o mesmo é usado, fato que dificulta o reuso de idéias. Este fato é destacado como barreira à engenharia de software .

Um antagonismo apontado pelo autor é que o fato de engenheiros de áreas diversas à ciência da computação desenvolverem programas sem a aplicação de técnicas adequadas, formais, conduz ao provável envelhecimento do software. Em contrapartida, engenheiros da ciência da computação entendem pouco do negócio.

Uma crítica aos pesquisadores da área adverte que estes devem repensar sua audiência com foco em quem realmente desenvolve sistemas e usará o conhecimento. É extremamente importante demonstrar os princípios da engenharia de software na prática, com produto real, viabilizando a verdadeira integração academia-mercado e a transferência de tecnologia.

Crítica

O texto, através de metáforas que associam o envelhecimento humano ao do software e causas e conseqüências do envelhecimento, apresenta coerência com o tema em discussão e bastante originalidade.

Embora o artigo tenha 14 anos, muitos dos aspectos apontados ainda são desafios no desenvolvimento de software. Como exemplo, a urgência na entrega de artefatos de software em prazos exíguos leva a negligência da documentação e da revisão. Também atual é a relevância da documentação e da revisão concomitantes ao projeto como alicerces para a manutenção do software e a garantia de sua longevidade.

O planejamento para a mudança é um aspecto não mais incipiente em pleno início do século XXI, quando se abordam técnicas de reuso como padrões de projeto e *frameworks*. A refatoração de código, um dos princípios das metodologias ágeis de desenvolvimento de software, tenta garantir que ocorra o rejuvenescimento constante daquele.

A dissociação entre a prevenção do envelhecimento e o tratamento deste é deveras interessante por valorar o software como investimento a ser preservado da deterioração, assim como os bens tangíveis. Não apenas software novo; a valoração do software legado é defendido e bem detalhado.

Uma conclusão interessante do Parnas suscita uma auto-crítica e é direcionada aos engenheiros de software, requerendo dos mesmos a ilustração dos princípios da área através de exemplos e modelos reais. Sua explanação sobre envelhecimento, suas metáforas e exemplos são uma materialização perfeita disso. Credito a isto a maior contribuição do artigo.

Questões para discussão

Como mensurar a longevidade de um software com projeto focado em mudança?
Que técnicas formais podem ser usadas num projeto robusto que suporte mudanças futuras com baixo custo?
Embora as Metodologias Ágeis defendam princípios semelhantes aos citados pelo Parnas quanto ao projeto focado em mudança, também defendem a simplicidade. Não seria antagônico um projeto mais simples possível suportar facilmente alterações futuras?
Como a refatoração de código, constante em metodologias como *Extreme Programming*, pode efetivamente garantir sua longevidade? Como mensurá-la? Como quantificar o ganho da revisão constante de código e documentação?
Tais métricas quantitativas são realmente relevantes ou o método empírico de observação é suficiente para comprovar as afirmações citadas pelo autor como fatores de deterioração de código?

Referência

PARNAS, D. L. Software Aging. In: 16th International Conference on Software Engineering, Sorrento, Itália, 1994, Proceedings… 1994, p. 279-287.