Engenharia de Software Fidedigno

Arndt von Staa
2006

1. Motivação

Em virtude da crescente participação de software na sociedade, torna-se cada vez mais necessário assegurar que seja fidedigno. Um software é dito **fidedigno** (inglês: *dependable*) quando se pode justificavelmente depender dele assumindo riscos de danos compatíveis com o serviço do software. Em [ALRL 2004] Avizienis e outros estabelecem as propriedades que sistemas fidedignos devem possuir. Com pequenas modificações temos:

Disponibilidade: estar pronto para prestar serviço correto sempre que se necessite do software.

Confiabilidade: habilidade de prestar continuamente serviço correto.

Segurança: (*safety*) habilidade de evitar conseqüências catastróficas relativas tanto aos usuários como ao ambiente.

Proteção: habilidade de evitar tentativas de agressão bem sucedidas.

Privacidade: habilidade de proteger dados e código contra acesso indevido.

Integridade: ausência de alterações não permitidas (corrupção de elementos).

Robustez: habilidade de detectar falhas de modo que os danos (as conseqüências de erros ou falhas) possam ser mantidas em um patamar aceitável.

Recuperabilidade: habilidade em ser rapidamente reposto em operação fidedigna após a ocorrência de uma falha.

Manutenibilidade: habilidade de ser modificado (evoluído) ou corrigido sem que novos problemas sejam inseridos.

Depurabilidade habilidade de apoio à diagnose e à eliminação de possíveis falhas a partir de relatos gerados.

Em [BB 2001] Basili e Boehm mencionam que cerca de 50% dos sistemas de software tornados disponíveis contêm faltas não triviais. Mencionam ainda que cerca de 90% do tempo inoperante de um sistema se deve a cerca de 10% das faltas nele contidas. Por outro lado, é sabido que falhas podem acontecer, independentemente de quão bem se tenha desenvolvido o software. As causas vão desde a reconhecida falibilidade humana até condições que fogem do controle dos desenvolvedores, tais como erros de máquina (possivelmente por causas externas), erros na plataforma usada (sistema operacional, hardware, rede, banco de dados, etc.), falta de energia, falhas em software que coexiste na máquina, agressões (ex. vírus, furto de dados, uso indevido).

Pode-se afirmar que é utópico almejar sistemas absolutamente fidedignos por construção [PBBCCCEFKMOSTTT 2002]. Consequentemente, como não se pode assegurar a ausência de falhas, deve-se tentar conviver com a presença delas através da construção de sistemas tolerantes a falhas, recuperáveis, diagnosticáveis e depuráveis. Uma possibilidade adicional

é permitir que sistemas possam reduzir a sua funcionalidade ou capacidade de modo a assegurar fidedignidade no evento de alguma falha [Bentley 2005].

Uma outra fonte de problemas é a deterioração do software [EKGMM 2001]. Diferente da manutenção de artefatos físicos, a deterioração do software tende a ser uma conseqüência da sua manutenção. Entretanto, é impossível evitar que ocorra a necessidade de evolução. Vários autores mencionam que 75% ou mais do esforço de manutenção decorre da evolução ou adaptação do software). Surge então a necessidade de desenvolver software de forma que seja manutenível. Ou seja, deseja-se que o custo da manutenção seja compatível com a dimensão da funcionalidade alterada e que se possa assegurar que a fidedignidade não seja comprometida à medida que se realizam manutenções. É reconhecido também que para tal é necessário que, à medida que o software vai sendo modificado, se deva realizar manutenção preventiva, ou seja reestruturar o software com vistas a restabelecer a necessária qualidade de engenharia.

2. Objetivos de longo prazo

É reconhecido ser um desafio desenvolver a baixo custo software com as características acima discutidas. Em particular deve-se levar em conta a grande tendência para o desenvolvimento de sistemas distribuídos (residentes na *Web*) e autônomos. É sabido que tais sistemas representam um desafio a todas as tentativas de garantia da fidedignidade tal como descrita acima.

Para alcançar estes objetivos é necessário aprimorar as ferramentas disponíveis. É necessário desenvolver *frameworks*, métodos, técnicas padrões de arquitetura e de projeto capazes de auxiliar os desenvolvedores a atingirem estes objetivos. Finalmente, é necessário reestruturar currículos e desenvolver material de ensino de modo que os profissionais formados estejam preparados a utilizar este novo instrumental.

3. Referências bibliográficas

[ALRL 2004] Avizienis, A.; Laprie, J-C.; Randell, B.; Landwehr, C.; "Basic Concepts and Taxonomy of Dependable and Secure Computing"; *IEEE Transactions on Dependable and Secure Computing* 1(1); Los Alamitos, CA: IEEE Computer Society; 2004; pags 11-33

[BB 2001] Basili, V.R.; Boehm, B.W.; "Software Defect Reduction Top 10 List"; *IEEE Computer* 34(1); Los Alamitos, CA: IEEE Computer Society; 2001; pags 135-137

[EKGMM 2001] Eick, S.G.; Karr, A.K.; Graves, T.L.; Marron, J.S.; Mockus, A.; "Does Code Decay? Assessing the Evidence from Change Management Data"; *IEEE Transactions on Software Engineering* 27(1); Los Alamitos, CA: IEEE Computer Society; 2001; pags 1-12

[PBBCCCEFKMOSTTT 2002] Patterson, D.; Brown, A.; Broadwell, P.; Candea, G.; Chen, M.; Cutler, J.; Enriquez, P.; Fox, A.; Kycyman, E.; Merzbacher, M.; Oppenheimer, D.; Sastry, N.; Tetzlaff, W.; Traupman, J.; Treuhaft, N.; *Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies*; Technical Report UCB/CSD-02-1175, Computer Science, University of California Berkeley; 2002; Buscado em: 7/8/2004; URL: http://roc.cs.berkeley.edu/papers/ROC_TR02-1175.pdf

[Bentley 2005] Bentley, P.; "Investigations into Graceful Degradation of Evolutionary Developmental Software"; *Natural Computing* 4(4); Berlin: Springer; 2005; pags 417-437

Arndt von Staa

Professor Associado do Departamento de Informática da PUC-Rio. É PhD em Ciência da Computação (1974, Engenharia de Software) pela Universidade de Waterloo, Ontário, Canadá. Atua em computação desde 1962. Suas atividades de pesquisa e desenvolvimento mais recentes concentram-se em técnicas de controle da qualidade de software, em métodos e processos de desenvolvimento de software de qualidade assegurada e em ambientes de desenvolvimento de software assistidos por computador. Idealizou, projetou e conduziu o desenvolvimento de TALISMAN, um meta-ambiente mono-usuário de engenharia de software assistido por computador (concluído em 1994). Assessora a incubadora Instituto Genesis da PUC-Rio em processos e tecnologias de Engenharia de Software. Publicou mais de 50 trabalhos entre livros, capítulos de livros, artigos em periódicos e artigos em conferências com revisão formal. Orientou 6 teses de doutorado e 52 dissertações de mestrado.