ANÁLISE DE RISCO – UMA PROPOSTA DE JUSTIFICATIVA PARA ADOÇÃO DE TÉCNICAS COMO A REFATORAÇÃO

Marcelo Marcilio Silva

Centro Estadual de Educação Tecnológica Paula Souza (CEETEPS)
CEP 01124-060 – São Paulo – SP – Brazil

marmarcilio@uol.com.br

Maurício Amaral de Almeida

Centro Estadual de Educação Tecnológica Paula Souza (CEETEPS)
CEP 01124-060 – São Paulo – SP – Brazil

madealmeida@gmail.com

RESUMO: O presente artigo tem por objetivo apresentar uma justificativa de negócio (mitigação de riscos) para adoção de práticas de desenvolvimento de sistemas que visam exclusivamente a melhoria de atributos não funcionais (manutenibilidade, legibilidade, redução de complexidade) dos sistemas de informação. São caracterizadas práticas como refatoração de sistemas, conceitos como complexidade de sistema e métricas de avaliação de complexidades e risco. É estabelecida uma relação entre o aumento da complexidade e o aumento de risco associado ao sistema, fato que per si, justificaria a adoção de práticas minimizadoras desse risco.

Palavras Chaves: Qualidade de Software, Risco, Gestão de TI

1. Introdução

Dos recursos atualmente empregados em Tecnologia de Informação (TI), a maior parte é consumida na administração de sistemas já desenvolvidos - sistemas legados, enquanto que outra pequena parte é empregada no desenvolvimento de novos projetos. As estimativas variam bastante, os mais otimistas arriscam percentuais em torno de 55% enquanto os mais pessimistas sugerem que os gastos com sistemas legados atinjam percentuais superiores a 80% [1]. De qualquer forma, o volume de recursos empregado na gestão dos sistemas legados é grande o suficiente para torná-lo um foco interessante ao pesquisador comprometido em melhorar a eficiência da gestão dos recursos investidos em TI.

2. Sistemas Legados – Caracterização

Neste documento, caracteriza-se como sistema legado, o sistema computacional já desenvolvido, provedor de serviços essenciais à organização.

Na literatura encontram-se várias observações sobre as dificuldades encontradas na gestão desses sistemas [2] [3] [4], no entanto, pode-se agrupá-las nas seguintes áreas:

- Complexidade de sistema – ocasionada por sistemas com elevado número de linhas (centenas de milhares), lógica de difícil compreensão seja pela extensão, dificuldade do problema ou necessidade de informação extra-código, elevado número de componentes;

- Dificuldade de manutenção – normalmente gerada pela falta de recursos para manutenção, pessoal, documental, hardware ou softwares envolvidos no processo de manutenção;

- Obsolescência tecnológica – oriunda das constantes mudanças tecnológicas sofridas na área de TI (hardware, software ou tendências);

- Desalinhamento entre as funções do sistema e as necessidades do negócio – propiciado pela constantes mudanças de mercado a que as organizações estão continuamente expostas.

As dificuldades acima apresentadas impactam diretamente na operacionalização dos sistemas legados e na performance das organizações como um todo. A superação destas dificuldades deve ser um foco constante das equipes envolvidas na gestão dos recursos de TI.

Sobre a atuação das equipes gestoras é conveniente ressaltar que estas podem apenas reagir de forma limitada a questões como o desalinhamento funcional e obsolescência tecnológica, pois tem pouco ou nenhum controle sobre as macrovariáveis envolvidas nessas mudanças; em contrapartida estas mesmas equipes podem assumir uma postura pró-ativa no tratamento de questões como a complexidade e a dificuldade de manutenção. Em função dessas limitações, concentrar-se-á a discussão deste trabalho nas atividades relacionadas à superação das dificuldades relacionadas ao controle da complexidade e facilitação da manutenção.

Como os sistemas legados oferecem serviços essenciais as organizações, dois importantes conceitos: disponibilidade e manutenibilidade de sistema, podem ser utilizados para mensurar a qualidade dos sistemas geridos. Para efeito de conceituação seguem algumas definições:

- disponibilidade – probabilidade de que um sistema execute funções pretendidas durante um período específico sob determinadas condições. Associa-se a este conceito a função:

  $D(t) = \int_t^{\infty} f(x)dx$, onde:

t – tempo até a ocorrência de falha

f(x) é a função densidade de probabilidade de falha.

Existe a necessidade de um esforço empírico para a formulação da função densidade de probabilidade. Apesar disso, a determinação de do MTBF (Mean Time Before Fail / Tempo Médio Antes de Falha) pode servir de orientação sobre o comportamento da disponibilidade;

- manutenibilidade – probabilidade de executar uma ação reparatória bem sucedida dado um intervalo de tempo. Associa-se a este conceito a função de probabilidade:

$M(t) = 1 - e^{-\mu.t}$, onde:

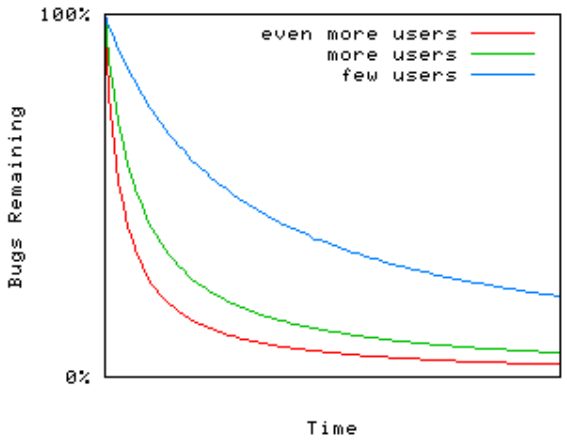
t – tempo de reparo;

$\mu = \frac{1}{MTTR}$, MTTR (Mean Time To Repair ou Tempo Médio Para Reparo)

Ambos os conceitos disponibilidade e manutenibilidade são aceitos e mencionados por metodologias de modelo de maturidade em desenvolvimento de software bem como normas internacionais como a ISO/IEC 9126 e SAE JA1002 e JA1003.

Se por um lado, os sistemas legados tendem a oferecer uma redução gradativa das falhas em função do tempo de utilização o que impacta positivamente no MTBF. A Figura 1 ilustra a diminuição de falhas encontradas nos sistemas em função do tempo de existência do sistema.

GRÁFICO 1 – número de falhas (Bugs) em função do tempo de uso



Fonte: DREPPER, 2005. [5]

Por outro lado, é perceptível que as novas implementações, sejam elas corretivas – para corrigir defeitos, ou aditivas – para atender a novas necessidades, tornam os sistemas maiores e mais complexos. Esse incremento afeta diretamente a extensão dos sistemas, bem como sua complexidade. Desta forma, é esperado um incremento do MTTR, o que resulta negativamente na manutenibilidade do sistema implicando num maior risco. Segundo Banker: *Com base em análise de projetos de manutenção de software, nós confirmamos que os custos de manutenção crescem significativamente quando a complexidade aumenta*[1] [6]. Normalmente, aumento de custos em TI estão associados a maiores aumento de prazo e ao risco de abandono de atividades [7].

3. Complexidade – Caracterização

Complexidade pode ser definida como:

> *grau de dificuldade em se compreender ou verificar a especificação ou implementação de um sistema ou parte constituinte*.
>
> *IEEE Standard Computer Dictionary [8]*

Existem várias métricas disponíveis para aferição da complexidade de um sistema de software. Almeida [9] recolhe uma série de possibilidades para a realização de métricas em código: tamanho; presença de documentação; Halstead ; Cyclomatic complexity ; construções sintáticas; estrutura; orientadas a objetos; e estrutura de dados.

Almeida apresenta a existência de vários fatores a serem considerados quando da seleção de métricas, dentre eles: a facilidade de automação das métricas e familiaridade com as equipes desenvolvedoras. Abaixo segue algumas métricas consideradas adequadas para a aferição da complexidade de um sistema.

Complexidade Ciclomática de McCabe - CCMc

Foi inicialmente enunciada por Thomas McCabe em 1976, assumindo a representação do programa como um grafo composto de nós e arcos, como sendo:

M = A – N + X, onde:

M – complexidade ciclomática de McCabe;

---

[1] Tradução do autor. Original: *On the basis of an analysis of software maintenance projects in acommercial application environment we confirmed that software maintenance costs rise significandy as software complexity increases.*

A – número de arcos no programa;

N – número de nós (pontos de decisão);

X – quantidade de pontos de saída do programa;

Assumindo programas com um único ponto de saída, pode-se associar a CCMc a expressão:

CCMc = D +1; onde D é o número de pontos de decisão do programa.

Para ilustrar, a Tabela 1 apresenta uma associação entre a CCMc é o risco de manutenção de sistema.

TABELA 1 – Valores de CCMc e riscos associados

CCMc	**Risco Associado**
1-10	programa simples, sem muito risco
11-20	complexidade média, risco moderado
21-50	complexo, alto risco
51+	instável, altíssimo risco

Fonte: CHARNEY, 2005 . Adaptado [10].

Ponto de Função - PF

Definido por Allan Albrecht em 1979, baseia-se na mensuração de requisitos funcionais tais como: entradas, saídas, solicitações de intervenção do usuário, arquivos associados e interfaces externas. É um método bem definido testado e padronizado por mais de 25 anos de uso no mundo todo. Apesar disso, existem críticos que questionam que este método é mais apropriado para a mensuração do nível de esforço (recursos) necessário para a produção de um sistema do que para a medição da complexidade do mesmo.

Linhas de Código-Fonte - LCF

Uma das primeiras métricas utilizadas para predição da produtividade de programadores e dimensionamento de esforço de desenvolvimento. Consiste na contagem automática das linhas de código dos sistemas em questão. Somente para ilustração, segue na TABELA 2 uma estimativa de linhas de código associadas a grandes sistemas:

Existem ainda outras métricas, contudo não é objetivo deste texto a revisão exaustiva das métricas existentes, apenas ilustrar para o leitor as principais para formar o arcabouço conceitual necessário a compreensão da

importância das práticas redutoras da complexidade e consequentemente redutora de riscos.

TABELA 2 – Número de linhas de código-fonte (LCF) por sistema

Ano	Sistema	LCF (Milhões)
1993	Windows NT 3.1	4-5
1994	Windows NT 3.5	7-8
1996	Windows NT 4.0	11-12[1]
2000	Windows 2000	mais de 29
2001	Windows XP	40
2003	Windows Server 2003	50

4. Risco – Caracterização

Como descrito no Integrated Risk Management Framework:

> *Risco refere-se a incerteza que envolve eventos ou resultados futuros. Ele é a expressão do impacto potencial de um dado evento em influenciar os objetivos de uma organização.*
>
> Canadá, 2001 [11].

Sob esta ótica, o risco apóia-se no tripé:

- futuro, refere-se ao porvir;
- incerteza, pode ou não ocorrer;
- possibilidade de gestão, pode ser evitado ou mitigado;

Novamente, também na visão de risco deve-se concentrar nas atividades das equipes gestoras de sistemas legados relacionadas às dificuldades associadas à complexidade e melhoria da manutibilidade pois atendem aos 3 quesitos necessários para a gestão de risco.

5. Ações Possíveis para Reduzir a Complexidade – Refatoração

A seguir é apresentada uma definição do que vem a ser refatoração:

> *Refatorar é o processo de modificar um sistema de software de tal forma a não alterar o comportamento externo do código e ainda melhorar sua estrutura interna.*
>
> Martin Fowler, 1999 [12].

Em outras palavras o sistema aparentemente permanece o mesmo, ele foi alterado apenas no seu interior visando promover atributos não funcionais de software, como aumentar a legibilidade, manutenibilidade; reduzir a complexidade, ou reorganizar a arquitetura interna visando maior extensibilidade.

O conceito de refatoração é construído sobre a idéia de “code smell” – cheiro, um sintoma que “sinaliza suavemente” um problema maior, mais estrutural. Para os desenvolvedores de sistemas isso é natural, mas a questão central continua sendo:

Como justificar o emprego de recursos hoje baseado em algo vago como “cheiro”, tendo em vista que não será agregado nada de novo ao sistema?

A resposta está em olhar para o futuro. Refatorar é preparar-se para um futuro incerto, isto é, arriscado. Somente a análise dos riscos envolvidos com o fato de estar “fora de forma” poderá dar a verdadeira dimensão de qual o real valor de uma atividade como a refatoração.

Simulação do Impacto da Redução da Complexidade

Determinar o impacto de cada evento é uma tarefa difícil e própria de cada sistema e organização, que requer atenção especial da equipe gestora dos sistemas legados. De qualquer maneira, será aqui apresentada uma análise dos possíveis ganhos a serem obtidos em função da redução da complexidade a partir do seu reflexo em parâmetros como o MTTR.

Para esse exercício é necessário assumir algumas posições que serão explicitadas a seguir:

- relação linear entre métricas de complexidade como LCF e PF e o MTTR, isto é, que um código com o dobro de linhas (LCF) ou de requisitos funcionais (PF), leve o dobro de tempo para ser corrigido, caso isso seja necessário. Em sistemas legados, esta é uma possibilidade real pois normalmente este tipo de sistemas tem boa parte do código implementada e não mais utilizada;
- a redução de métricas como o CCMc apresenta resultado superior ao linear. Desta forma é possível supor ganhos superiores aos apresentados;

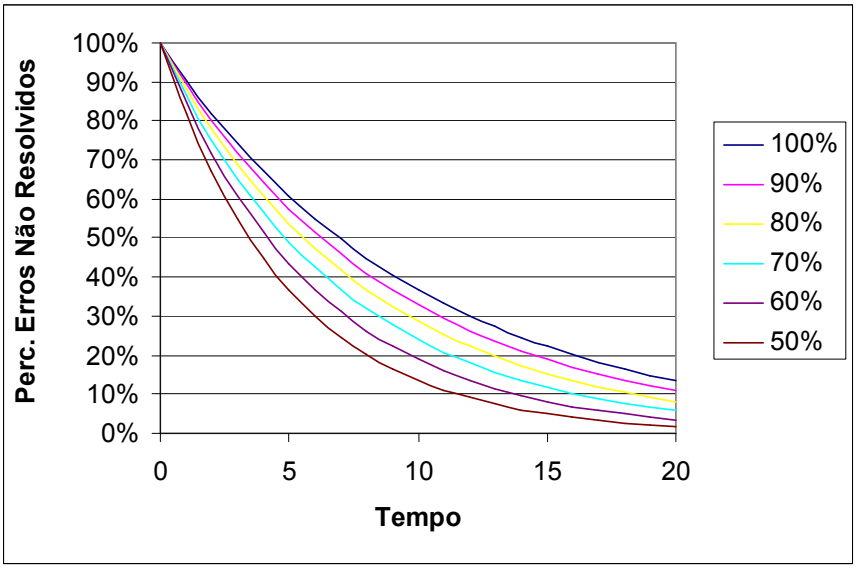
Com base nessas premissas podemos verificar um caso de estudo com MTTR inicial de 10h e as referidas curvas de diminuição de complexidade até 50% da inicial. O GRÁFICO 2 exibe esse caso particular.

É possível observar que a redução da complexidade permite uma resposta muito mais ágil em relação ao número de problemas resolvidos

aumentando significativamente o tempo do sistema em plena condição de uso e consequentemente as taxas de satisfação dos usuários.

A análise de riscos deve considerar não apenas os ganhos advindos da melhora dos atributos não funcionais do software como também os impactos gerados pela paralisação mesmo que momentânea dos sistemas. Estes impactos normalmente implicam em lucros cessantes que não raras vezes, perfazem somas muito superiores aos custos de qualquer atividade de refatoração.

GRÁFICO 2 – Percentual de problemas não reparados em função do tempo



6. Conclusão

Como foi visto é possível justificar a adoção de medidas, como a refatoração, para a melhoria de atributos não funcionais dos sistemas legados devidamente apoiados no ferramental para a gestão dos riscos e dimensionamento dos possíveis impactos gerados pelos eventos associados. Esse arcabouço teórico tem sido desenvolvido por vários institutos como o PMI e o National Institute of Science and Technology e está também refletido em normas de engenharia de risco como a ISO/DIS 31000.

Como já foi dito, refatorar é preparar-se para o futuro, é inovar, mas para que esse processo de inovação seja bem sucedido é fundamental que ele esteja bem definido, seja viável, e possa contar com os recursos necessários à sua conclusão.

6. Referências Bibliográficas.

[1] Benson, R., Bugnitz, T., Walton, W. (2004), *From Business Strategy to IT Action – Right Decisions for a Better Bottom Line*. pg. 1-30. New Jersey: John Wiley & Sons.

[2] RAMOS, C. S., OLIVEIRA, K. M., ANQUETIL, N. (2010), *Conhecendo Sistemas Legados através de Métricas de Software.*Disponível em <www.sbc.org.br/bibliotecadigital/download.php? paper=258>. Acesso em: 30 Mai 2010.

[3] SILVA, L. de P., SANTANDER, V. F. A. (2010), *Uma Proposta de Evolução em Sistemas Legados.* Univ. Est. Oeste do Paraná. Disponível em: <http://wer.inf.puc-rio.br/WERpapers/ artigos/artigos WER04/ Luciana Paiva.pdf>. Acesso em: 02 Jun 2010.

[4] MIRANDA, M. A., BRICK, E. S. (2003), *Modelo de confiabilidade, disponibilidade e manutenibilidade de sistemas, aplicado a plataformas de petróleo.* Univ. de Ouro Preto, 2003. Disponível em <http://www.abepro.org.br/biblioteca/ENEGEP2003_ TR0103_0030.pdf>. Acesso em: 02 Jun 2010.

[5] DREPPER, U. (2010). *Users and Complexity vs Bugs.*Disponível em: <http://udrepper. livejournal.com/6079.html>. Acesso em: 28 Mai 2010.

[6] Banker, R. D., Datar, S. M., Kemerer, C. F., Zweig, D. S. (1990), *Software Complexity and Software Maintenance Costs*. School of Management. MIT.

[7] McManus, J., Wood-Harper, T. (2007), "Understanding the Sources of Information Systems Project Failure". *Management Services, Autumn 2007.*

[8] IEEE. (1990), *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. New York, NY.

[9] ALMEIDA, M. A. (1999), *Geração de Modelos de Qualidade de Software através da Abordagem de Aprendizagem Automática*. Tese de Doutoramento em Engenharia Elétrica. Universidade de São Paulo.

[10] CHARNEY, R. (2005) *Programming Tools: Code Complexity Metrics*. Linux Journal, 2005. Disponível em <http://www.linuxjournal.com/article/ 8035>. Acesso em: 03 Jun 2010.

[11] CANADÁ (2009). *Guide to Using the Project Complexity and Risk Assessment Tool.* Treasury Board of Canada Secretariat. CANADÁ, 2009. Disponível em <http://www.tbs-sct.gc.ca/pm-gp/doc/pcrag-ecrpg/pcrag-ecrpg-006-eng.aspx>. Acesso em: 03 Jun 2010.

[12] Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D. (1999), *Refactoring: Improving the Design of Existing Code*. Addison Wesley.