A Influência de Arquiteturas Abertas em Sistemas de Defesa para Obtenção de Requisitos de Dependabilidade

Alexandre de Barros Barreto e Edgar Toshiro Yano
Div de Ciência da Computação
Instituto Tecnológico de Aeronáutica
S.J.Campos, S. Paulo, Brazil
e-mail: kabart@gmail.com/ yano@ita.br

***Resumo*—Sistemas de Defesa diferenciam-se de sistemas comuns pelas conseqüências catastróficas quando da existência de uma falha grave (perdas humanas e/ou falha no cumprimento da missão). Nesse tipo de sistema, deve-se garantir tanto os requisitos funcionais, como os relacionados ao ambiente onde o sistema irá operar, o que é chamado de Dependabilidade. A presente pesquisa apresenta uma análise quantitativa entre como as arquiteturas abertas e fechadas influenciam na obtenção de um maior grau de dependabilidade. Para isso um modelo matemático foi usado como base para a supracitada avaliação. Ao final pode-se concluir que em sistemas de defesa deve-se preferencialmente usar arquiteturas abertas, visando dessa forma aumentar o seu grau de dependabilidade.**

***Palavras-chaves*—Dependabilidade, Arquiteturas Abertas e Fechadas.**

I. Introdução

A sociedade moderna possui como um dos fatores básicos para seu funcionamento o uso intensivo de sistemas computacionais, onde [1] cita que para analisar uma sociedade moderna deve-se verificar abundância dos recursos computacionais (richness), o grau de dependência da sociedade a ela, bem como a relação entre as duas. Esse fato é simples de ser observado, bastando para isso a verificação do grau de dependência de vários sistemas críticos (metrô, tráfego aéreo, hidrelétricas, etc).

Uma das áreas que mais fazem uso de sistemas computacionais é a defesa. Uma característica marcante desse tipo de sistema é que na ocorrência de uma falha, os resultados podem ser catastróficos. Como exemplo, pode-se citar a queda de um avião F-18 devido a um erro de lógica existente em seu código fonte [2], onde além da perda material envolvida, ainda existe a questão do comprometimento da missão desempenhada pela aeronave [3].

Sendo assim, além da necessidade de verificação e validação dos requisitos funcionais, ou seja, àqueles que definem *o que* deve ser feito pelo sistema, se faz necessário, também, garantir os relacionados ao seu ambiente de funcionamento, que a literatura classifica como requisitos qualitativos [4].

Até 1985, os requisitos qualitativos foram tratados de forma secundária, quando [5] introduziu o conceito de dependabilidade de sistemas, que consiste da propriedade na qual se garante que um software é confiável o suficiente para possibilitar o seu uso .

Apesar da reconhecida importância dos requisitos de dependabilidade para a aceitação de um sistema, ainda está em aberto a discussão sobre como os mesmos podem ser obtidos, pois uma vez que é impossível detectar todos os erros existentes em um software [6], é necessário identificar quais requisitos são os mais críticos, ou seja, àqueles que podem gerar resultados catastróficos.

Durante quase 20 anos, diversos estudos buscaram como minimizar a existência de erros em sistemas críticos. Porém, somente em 2006, o Departamento de Defesa Americano (DoD) apresentou uma possível resposta para questão [7], que consistia na adoção preferencial de arquiteturas abertas para o desenvolvimento de seus sistemas de defesa. O supracitado trabalho teve como escopo de análise como os atributos temporais e financeiros influenciavam o desenvolvimentos de sistemas bélicos, deixando de fora como esses sistemas influenciavam na obtenção dos requisitos de dependabilidade.

O presente trabalho tem como objetivo apresentar como as características existentes nas arquiteturas abertas podem simplificar o processo de obtenção dos requisitos de dependabilidade. Para isso foram usadas duas hipóteses na sua construção.A primeira hipótese diz que o uso de arquiteturas abertas, em virtude de suas características, aumenta a capacidade de prevenir a existência de defeitos, aumentando conseqüentemente a dependabilidade de sistemas de defesa quando comparadas com arquiteturas fechadas.

A segunda hipótese diz que em virtude de possuírem um número maior de usuários e uma maior diversidade de aplicações, os sistemas abertos podem ser mais ex-

austivamente testados, conduzindo então para um maior grau de dependabilidade.

Apesar de ser intuitivo a vantagem do uso de arquiteturas abertas em detrimento das fechadas, quando o foco é a questão da obtenção de dependabilidade, a presente pesquisa utilizou uma abordagem mais formal para validar as hipóteses ora apresentadas. Para tal, foi utilizada o modelo apresentado por [8]. Nesse trabalho os principais componentes computacionais usados em um sistema de defesa foram avaliados usando uma solução aberta e fechada. Ao final pode-se concluir que os sistemas abertos apresentam uma grande vantagem sobre os fechados no que tange aos aspectos de dependabilidade.

O presente documento está organizado em quatro seções. Na primeira será realizada uma presente introdução sobre o problema, bem como o objetivo do presente trabalho. Na sequência, será apresentada uma breve revisão bibliográfica. A seção 3 apresentará o método utilizado, bem como o experimento, o resultado e uma análise sobre o mesmo. Para finalizar será realizada uma breve conclusão, onde além de algumas considerações finais serem traçadas, alguns trabalhos futuros serão elencados.

II. Revisão Bibliográfica

A. Conceitos Básicos de Dependabilidade

De acordo com a definição apresentada em [9], a dependabilidade de um sistema computacional consiste em sua capacidade de fornecer um serviço considerado confiável, ou seja, sem a presença de defeitos.

Segundo [10], um defeito consiste em qualquer comportamento do sistema que se desvia da sua especificação. Segundo o mesmo autor, os defeitos podem ser classificados em quatro tipos:

- **Falha (fault)**: consiste em uma anomalia do sistema que pode vir a reduzir seu desempenho, porém não pode ser mensurada e observada;
- **Erro**:São resultados (observáveis e mensuráveis) da exploração de defeitos de um sistema, porém não ocasionam perdas de desempenho;
- **Degradação (degradation)**:São resultados da exploração de defeitos de um sistema, que acarretam perdas de desempenho do mesmo; e
- **Failures**: É a incapacidade de o sistema em desenvolver alguma de suas funcionalidades pelo qual foi projetado devido a um defeito explorado.

Uma vez entendido o conceito de defeito, é necessário evitar que eles não afetem o sistema alvo, ou seja, garantir que os requisitos de dependabilidade sejam obtidos. As técnicas comumente usadas para a obtenção da dependabilidade são: *prevenção de falhas (fault prevention)*, *tolerância à falhas (fault tolerance)*, *remoção das falhas (fault removal)* ou *previsão da falha (fault forecasting)*.

A técnica de prevenção de falhas consiste em garantir, através de estratégias de controle de qualidade durante o projeto e desenvolvimento dos hardwares e softwares, que o mesmo estará livre de falhas críticas durante seu funcionamento.

Por tolerância a falhas, entende-se o conjunto de ações necessárias para manter o sistema funcionando mesmo quando na presença de falhas. A implantação dessa técnica consiste em criar estruturas redundantes, que na existência de uma falha, entra em atividade.

A técnica de previsão de falhas consiste em usar um módulo de monitoramento, cuja tarefa é identificar comportamentos que sejam caracterizados como erro e, antes mesmo desse defeito reduzir o desempenho do ambiente computacional, um segundo módulo, através de técnicas de inteligência artificial, eliminam os fatores geradores da falha.

Por fim, a técnica de remoção de falhas consiste na atividade de eliminação de defeitos encontrados nos sistemas, durante a fase de projeto e manutenção, mesmos aqueles que foram projetados usando a prevenção de falhas, uma vez que a existência de erros é normal [11].

Apesar de todas as técnicas serem importantes para a obtenção da dependabilidade, no presente documento serão apresentadas apenas as de prevenção e remoção de falhas, pois as demais podem ser igualmente obtidas em sistemas abertos e fechados, sendo uma questão de definição da arquitetura a ser usada.

De posse da constatação de [11] e [6] sobre a existência permanente de falhas em sistemas computacionais, cabe ao arquiteto de sistemas a tarefa de descobrir e eliminar, durante a fase de análise e projeto, àqueles que possam vir a comprometer o funcionamento do mesmo, ou seja, os defeitos do tipo *failures*.

Na Engenharia de Sistemas essa tarefa é realizada através de duas técnicas: a validação e a verificação. A verificação consiste na técnica de testes cujo objetivo é identificar se o sistema construído/adquirido cumpre a especificação definida, ou seja se o sistema implementa os requisitos previstos em sua especificação. Enquanto a validação tem por finalidade checar se o sistema está implementado de forma correta, ou seja se os requisitos foram devidamente delineados.

Para realizar a tarefa de verificação e validação podem-se usar duas técnicas de testes: os de caixa branca e os de caixa preta. Os testes de caixa branca, também chamadados de testes estruturais, avaliam o comportamento interno do componente de software ou hardware. Essa técnica trabalha diretamente sobre o código-fonte do componente para avaliar aspectos como: condição, fluxo de dados, ciclos e caminhos lógicos. Os aspectos avaliados nesta técnica dependerão da complexidade e

da tecnologia utilizada na construção do componente.

Por sua vez, o teste de caixa preta, também chamado de teste funcional, é aquele onde o componente de software/hardware a ser testado é analisado do ponto de vista das entradas e saídas geradas pelo mesmo, ou seja, não são considerados os seus estados internos. O componente a ser testado pode ser um método, uma função interna, um programa, um componente, um conjunto de programas e/ou conjunto de componentes.

Tanto a verificação, como a validação podem usar os testes de caixa branca e/ou preta no seu desenvolvimento, devendo-se apenas se ressaltar a limitação dos testes de caixa branca em sistemas fechados. Porém em ambos os casos, o aspecto mais complexo está em definir o que deve ser testado, pois o conhecimento prévio e a experiência do analista serão vitais para o uso dessas ferramentas.

Uma vez que os testes identificam apenas a presença de falhas e não a sua ausência [6], quanto maior for a amostra de software e hardware analisados (amplitude/cobertura do teste), maior será a probabilidade de terem sido encontrados todos os defeitos e vulnerabilidades existentes no mesmo, sendo necessário então o uso de conceitos estatísticos [4] para definir o que deve ser testado.

B. *Arquiteturas Abertas*

Uma segunda definição para esse termo pode ser visto em [7] e [16], onde eles incluem na definição o conceito do processo de desenvolvimento para considerar uma arquitetura como aberta. Nessas abordagens para que que o sistema possa ser aberto, ele deve obrigatoriamente usar: padrões abertos e interfaces (conforme definição inicialmente apresentada), sistemas abertos [15], tecnologia de desenvolvimento colaborativo e ágil.

Uma característica marcante em sistemas abertos é que, devido ao acesso ao código ser garantido, muitos usuários contribuem durante o seu processo de desenvolvimento, realizando a sua revisão e documentação, aumentando a confiança dos usuários em relação a inexistência de portas traseiras e outros problemas de segurança [18].

Uma questão importante para o desenvolvimento de sistemas críticos e de defesa, muitas vezes abordados de forma parcial por alguns autores [7], é que as arquiteturas abertas diminuem a segurança de sistemas críticos, uma vez que o prévio conhecimento de detalhes de seu funcionamento dá aos atacantes a capacidade de planejar ataques, através das descobertas de falhas e vulnerabilidades.

Isso é uma verdade parcial, pois a obscuridade deve ser negada sempre. A obscuridade dá a falsa impressão de segurança ao usuário, pois sempre existirão métodos de corromper um sistema e seu prévio conhecimento não garante a descoberta desses meios [19]. Um exemplo seria uma carta armazenada em um cofre, o fato de o projeto de desenvolvimento do cofre ser público, não garante a um atacante a capacidade de arrombá-lo.

Apesar disso, em alguns casos o uso de obscuridade pode ser desejável. Nesse caso, deve-se analisar quais partes do projeto possuem classificação e, através dos conceitos de reuso, usar arquiteturas abertas onde é possível [20].

III. Experimento e Análise

A. *Método de Avaliação*

Conforme apresentado anteriormente, no que diz respeito às técnicas de prevenção e remoção da falha, uma série de fatores influenciam direta ou indiretamente na obtenção dos requisitos de dependabilidade de um sistema. Também foi apresentado que um dos maiores problemas consiste em identificar o que testar, sendo usado para isso método estatísticos.

No presente trabalho, será utilizado o método apresentado em [8], que define que a dependabilidade é obtida através de dois diferentes argumentos. No primeiro, quanto maior a quantidade de usuários, maior será a cobertura de testes a ser realizado. Isto é simples de ser verificado, pois o processo de obtenção da dependabilidade (garantia da qualidade) é uma atividade que não termina após a entrega do sistema, mas sim durante todo o seu ciclo de vida [4].

O segundo argumento utilizado define que quanto maior a diversidade de usuários, o que o autor chama de perfil, maior será a qualidade dos testes a serem realizados. Esse fato também é facilmente verificado, pois se os usuários são diferentes, é natural que eles realizem testes de funcionalidades diferentes em um mesmo sistema.

Um fato importante no supracitado modelo é que ele não diferencia usuários comuns de desenvolvedores, onde ele considera isso uma diversidade do tipo de usuário.

Para um entendimento do modelo de [8], é necessário que algumas variáveis sejam definidas:

- **Demanda**: quantidade de entradas possíveis de serem implementadas pelo usuário, onde pode-se usar para isso a complexidade do sistema em voga como medida de demanda [21].
- q_{ij}:probabilidade condicional de uma falha *i* ser causada por um usuário *j*.
- r_{ij}:probabilidade condicional de uma *failure* (impossibilidade do sistema cumprir com um ou mais requisitos pelo qual foi planejado) causado por uma falha *i* ser reportado por um usuário *j*.
- f_j: probabilidade condicional de um failure ser resolvido, uma vez reportado.
- T_j:número de demandas aplicadas pelo usuário. Essa variável está diretamente relacionado ao número total de usuários.

Segundo o supracitado modelo, a probabilidade de uma falha ser corrigida ($Pf_{(falhacorrigida)}$) é definida pela expressão 1 e 2, enquanto a probabilidade final de uma falha em um sistema com base em sua demanda (*pfdf*) é definida em 3.

$$Pf_{(FalhaNãoReportada)} = \prod_{k \in usuários} (1 - r_{ik} * q_{ik})^{T_k} \quad (1)$$

$$Pf_{(FalhaCorrigida)} = 1 - Pf_{(FalhaNãoReportada)} \quad (2)$$

$$P_{fdf} = \sum_{i \in regiõesdefalhas} (q_{ij} * Pf_{(FalhaCorrigida)}) \quad (3)$$

De posse das equações contidas em 1, 2 e 3, pode-se perceber alguns fatos:

- Quanto maior o número de usuários, maior será a capacidade de um sistema ter uma falha reportada.
- A complexidade do sistema é apresentada pela variável T_k.
- A variável *i* existente no somatório de *pfdf* (3) define o diversidade de usuários, uma vez que regiões de falhas diferentes, requerem usuários diferentes.
- Quanto maior for o valor da variável *pfdf*, maior será o grau de dependabilidade do sistema em voga.

B. Cenário de Estudo

Para avaliar se o grau de dependabilidade de um sistema possa ser influenciada pela adoção de uma arquitetura aberta ou não, será usado o método de [8] para comparar as duas soluções.

Uma vez que todos os sitemas de defesa necessitam de um sistema operacional, como restrição de escopo, visando simplificar a análise, será aplicado o supracitado método para comparar 02 sistemas operacionais (abertos e fechados), de forma ao final refutar ou não as hipóteses levantadas no presente artigo.

Apesar do método apresentado em [8] definir que aspectos devem ser levados em consideração na avaliação do grau de dependabilidade de um sistema, uma abordagem mais prática de como obter estes indicadores se faz necessário.

É necessário que se realize uma crítica ao supracitado modelo, pois ele considera usuários comuns e desenvolvedores da mesma forma. Usando a abordagem feita por [22], chega-se a uma conclusão diferente. Segundo ele, a probabilidade de um erro ser descoberto e, consequentemente, reportado (rij) é diferente com base no tipo de usuário onde a técnica é aplicada.

Segundo [22], o valor para r_{ij} quando é usado uma técnica formal de descoberta de falhas é de até 90%, enquanto o seu oposto chega a no máximo 50%, onde esse número pode ser considerado os erros detectados pelos usuários comuns. Esse indicador demonstra que os erros detectados por usuários é aproximadamente 50% menor que o encontrado pelos desenvolvedores. Dessa forma o índice r_{ij} será multiplicado por um fator de conversão que é a quantidade de desenvolvedores, conforme definido na tabela I.

Tabela I
NÚMERO DE DESENVOLVEDORES

Sistema Operacional	**Número de Desenvolvedores**	**rij**
Windows 7	1.000 [23]	62,00%
Debian 5	1.650 [25]	48,00%

O segundo aspecto que deve ser definido é o qij, ou seja, a probabilidade de uma falha i ser causada por um usuário j. Essa variável está diretamente relacionado ao número de usuários do sistema. Para definir esse número foi usada a tabela II.

Tabela II
NÚMERO DE USUÁRIOS

Tipo de Dispositivos	Servidores	
	Linux	Proprietário
Dispositivos Móveis [27]	26,60%	70,90%
Servidores [26]	63,70%	36,40%
qij	45,15%	53,65%

Por fim, deve-se definir qual será o índice usado para avaliar o grau de diversidade do sistema, variável *i* e variável T_k. Observe que nos dois casos, uma maior diversidade de usuários definirá um maior grau para o resultado final, diferenciando-se apenas no grau de influência sobre o resultado. Enquanto T_k incrementa de forma exponencial o resultado, o índice *i* influencia de forma multiplicativa. Dessa forma, as duas variáveis serão definidas com base na quantidade de diferentes aplicações em que elas são usadas, o que é apresentado na tabela III.

Tabela III
DIVERSIDADE DE USUÁRIOS [28]

Sistema Operacional	Total de Dispositivos	$n = i = T_k$
Proprietário	202	61,00%
Opensource	324	49,00%

C. Análise do Resultado

Aplicando os dados coletados nas equações apresentadas em 1, 2 e 3 chega-se nos resultados existentes na tabela IV.

De posse dos resultados supracitados, é simples observar que as arquiteturas abertas tem capacidade de gerar um índice de dependabilidade maior que as fechadas, o que prova a primeira hipótese levantada.

Tabela IV
AVALIAÇÃO DA DEPENDABILIDADE

Sistema Operacional	Sistema Fechado	Sistema Aberto
r_{ij}	0,48	0,62
q_{ij}	0,53	0,45
n	0,49	0,61
$Pf_{\{falhanoreportada\}}$	0,9266	0,9141
$Pf_{\{falhacorrigida\}}$	0,0734	0,0859
p_{fdf}	0,0191	0,0236

Ao analisar cada um dos índices de forma individual, nota-se a importância da diversidade das plataformas na obtenção do índice *pfdf*. Esse fato prova a segunda hipótese levantada.

Dessa forma pode-se concluir parcialmente que sistemas de defesa devem ser construídos preferencialmente usando arquiteturas abertas, onde o termo parcial, deve-se ao fato de não ter sido realizada uma análise em todas as técnicas existentes para a obtenção da dependabilidade, nem ter sido possível o acesso a um volume maior de dados.

IV. CONCLUSÃO

Conforme apresentado no decorrer do presente artigo, a construção de um sistema com alto grau de dependabilidade é uma atividade complexa, sendo necessário garantir que ele não apresente defeitos graves em todo o seu ciclo de vida. Em sistemas de defesa e de aplicação críticaa a ocorrência de erros pode ter conseqüências catastróficas, o que faz com que a observância dos requisitos de dependabilidade tenha que ser mais rígido.

O presente trabalho demonstrou que sistemas de defesa devem ser construídos preferencialmente com o uso de arquiteturas abertas, uma vez que estas, quando comparadas às fechadas, possuem características que potencializam a obtenção dos requisitos de dependabilidade.

Apesar dessa conclusão poder ser obtida de forma intuitiva, uma análise formal [8] foi realizada, tornando os resultados difíceis de serem refutados.

Ao final do trabalho pode-se concluir que o uso de sistemas abertos pode levar a obtenção de um grau mais alto de dependabilidade em sistemas de defesa.

Como trabalhos futuros, cita-se a expansão do presente trabalho, através da análise de outros componentes que fazem parte de um sistema computacional crítico, como compiladores, bibliotecas de precisão matemática e bibliotecas de criptografia. Esse estudo garantirá uma comparação mais completa da aplicação desse modelo em sistemas de defesa.

REFERÊNCIAS

[1] Evans, P.,Wurster, T.: Blown to Bits. Harvard Business School Press (2000).

[2] Efremides, S. Self Healing Systems. Athens Information Technology, 2005.

[3] Alberts, David S., John J. Gartska, and Frederick P. Stein. Network Centric Warfare, Developing and Leveraging Information Superiority (2nd Edition, Revised). CCRP publication series, August 1999/Second printing February 2000.

[4] Roger S. Pressman. Software Engineering - A Practioner's Aproach. The MacGraw-Hill, 4th. Edition.

[5] Laprie, J.C. Dependable computing and fault-tolerance: concepts and terminology. Digest of FTCS-15, p. 2-11, jun. 1985.

[6] Dijkstra, E.W., "Structured Programming,"Software Engineering Techniques, Buxton, J.N., and Randell, B., eds. Brussels, Belgium, NATO Science Committee, 1969.

[7] Payton, S. Open Technology Development ? Roadmap Plan. Washington: Secretary of Defense Advanced Systems & Concepts, 2006.

[8] D. Bosio; B. Littlewood; L. Strigini; M. J. Newby. Advantages of open source process for reliability: clarifying the issues. Workshop on Open Source Software Development, Newcastle upon Tyne, February 2002.

[9] Avizienis, A; Laprie, J.C. e Randell, B. Dependability of Computer Systems: Fundamental concepts, terminology, and examples. Technical Report, LAAS-CNRS, October, 2000.

[10] Parhami, B. Defect, Fault, Error,?, or Failure? Santa Barbara: University of California, 1997.

[11] MYERS, G. The Art of Software Testing. Wiley, 1979.

[12] Hecht, H. System Reliability and Failure Prevention. Boston: Artech House, 2004.

[13] Schneier, B. Attack trees: Modeling security threats. Dr. Dobb's journal, December 1999.

[14] WEBOPEDIA. Disponível em: http://www.webopedia.com/TERM/O/open_architecture.html. Acesso em 05/07/2011.

[15] OSI, 2011: The Open Source Initiative: Open Source Definition. Disponível em: http://www.opensource.org/osd.html. Acesso em 10/07/2011.

[16] Oreizy, Peyman. Open Architecture Software: A Flexible Approach to Decentralized Software Evolution. University of California, 2000.

[17] Gacek,C.; Lawrie,T.; Arief,B. The many meanings of Open Source. Technical Report CS-TR-737, Department of Computing Science, University of Newcastle upon Tyne, August 2001.

[18] Herz, J.C. e Scott, J. COTR Warriors: Open Technologies and Business of War. The DoD SoftwareTech News, vol. 10 no. 2, jun. 2007.

[19] Stinson, D.R. Cryptography Theory and Practice. CRC Press, 2a. ed., fev. 2002.

[20] Weathersby, J.M. Open Source and Long Road to Sustainability within U.S. DoD IT System. The DoD SoftwareTech News, vol. 10 no. 2, jun. 2007.

[21] Hutcheson,Marnie L.. Software Testing Fundamentals. Wiley Publishing, 2003.

[22] Jones,C. Software Quality for 1995: What Works and Doesn't. Software Productivity Researchs (www.spr.com)

[23] Sinofsky, S.The Windows 7 Team. Disponível em: http://blogs.msdn.com/b/e7/archive/2008/08/ 18/windows_5f00_7_5f00_team.aspx. Acessado em: 10/07/2011.

[24] O'Brien, L. How Many Lines of Codes in Windows? Disponível em: http://www.knowing.net/index.php/2005/12/06/how-many-lines-of-code-in-windows/. Acessado em: 10/07/2011.

[25] Debian Developers Database. Disponível em: http://db.debian.org/. Acessado em: 07/07/2011.

[26] Usage of Unix for websites. W3Techs. Disponível em: http://w3techs.com/technologies/details/os-unix/all/all. Acessado em: 07/09/2010.

[27] Gartner Says Worldwide Mobile Phone Sales Grew 35 Percent in Third Quarter 2010; Smartphone Sales Increased 96 Percent. Disponível em:

http://www.gartner.com/it/page.jsp?id=1466313.Acessado em: 10/11/2010.

[28] Estimating the number of Linux users. Disponível em: http://counter.li.org/estimates.php. Acessado em: 10/07/2011.