Uma Visão Geral de Teste de Sistemas Embarcados Com Enfoque no Teste Estrutural

Vânia de Oliveira Neves[1]

[1] Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo – São Carlos, SP – Brasil

vaniaon@icmc.usp.br

Abstract. *Embedded systems are increasingly involved in day-to-day life of people. However, if these systems do not work as intended may cause serious damage, injuries, deaths or even disasters. Thus, the testing activity is essential in this area. This paper presents an overview of embedded systems as well as the challenges related to testing of these kind of systems. Are also presented four papers in the area of embedded system testing with focus on technique for structural testing.*

Resumo. *Sistemas embarcados estão cada vez mais presentes no dia-a-dia das pessoas. No entanto, caso eles não funcionem conforme pretendido poderão causar prejuízos graves, com ferimentos, mortes ou até catástrofes. Dessa forma, a atividade de teste é essencial nessa área. Neste artigo é apresentada uma visão geral de sistemas embarcados bem como dos desafios relacionados ao teste desses tipos de sistemas. Também são apresentados quatro trabalhos na área de teste de sistema embarcados com enfoque na técnica de teste estrutural.*

1. Introdução

Os sistemas embarcados estão presentes na vida das pessoas de tal forma que é difícil imaginar o dia-a-dia sem eles. Podem ser encontrados em sistemas médicos, no setor de aviação, automotivo, no uso doméstico, etc. Exemplos de sistemas embarcados incluem: sistemas de navegação, controle de injeção eletrônica, GPS, impressoras, câmeras digitais, marca-passo, fornos de microondas, celulares, vídeos-games, etc.
Segundo Ebert e Salecker (2009), o mercado mundial de sistemas embarcados é de aproximadamente 160 bilhões de euros, envolvendo cerca de 3 bilhões de unidades embarcadas entregues por ano e um crescimento anual de 9 por cento. Devido à natureza dos sistemas embarcados, a maioria das vendas é do lado do hardware, no entanto, a importância do software e serviços está aumentando rapidamente. Carros hoje tem 100 MB de software em execução, com uma complexidade crescendo cada vez mais rápida do que em sistemas tradicionais (Ebert e Salecker, 2009). A complexidade de software embarcado é bem maior se comparado com os softwares tradicionais, devido a características como tempo-real, limitações de interface, entre outras.

Segundo Tian et al (2009), uma tendência dos sistemas embarcados é que cada vez mais deixarão de ser definidos pelo hardware e passarão a ser projetados para fazer qualquer função que satisfaça objetivos múltiplos e mutáveis. Dessa forma, os sistemas embarcados envolvem muitos desafios que transcendem os requisitos tradicionais das aplicações. Entre esses desafios, está o teste de sistemas embarcados.

O objetivo deste artigo é apresentar uma visão geral de teste de sistemas embarcados, com enfoque no teste estrutural. Na seção 2 são discutidas algumas das características desses sistemas; na seção 3 é apresentada uma visão geral dos desafios/dificuldades no teste de tais sistemas; na seção 4 são discutidos alguns trabalhos que envolvem teste estrutural de sistemas embarcados e, por fim, na seção 5, são apresentadas as conclusões obtidas.

2. Sistemas Embarcados

Sistemas embarcados podem representam uma ampla gama de sistemas e, por isso, é possível encontrar na literatura várias definições diferentes para eles. Por exemplo:

- "Parte de um produto na qual o usuário final não interage diretamente ou controla" (Tian et al, 2009)
- "É qualquer sistema de software que é projetado em uma plataforma diferente daquela que é prevista dele ser implantado" (Berger, 2001)
- "São dispositivos utilizados para controlar, monitorar e apoiar o funcionamento dos equipamentos, máquinas ou instalações. São parte integrante de um sistema." (Ebert e Salecker ,2009)
- "Projetados para uma tarefa específica" (Ebert e Salecker ,2009)

No entanto, todos os sistemas embarcados apresentam uma característica comum: interagem com o mundo físico real, controlando algum hardware específico.

Um sistema embarcado interage com o mundo real, recebendo sinais através de sensores e enviando sinais de saída para atores que de alguma forma manipulam o ambiente. O ambiente de um sistema embarcado, incluindo atores e sensores, é muitas vezes chamado de planta (Broekman e Notenboom, 2002). A Figura 1 apresenta um esquema genérico de um sistema embarcado:

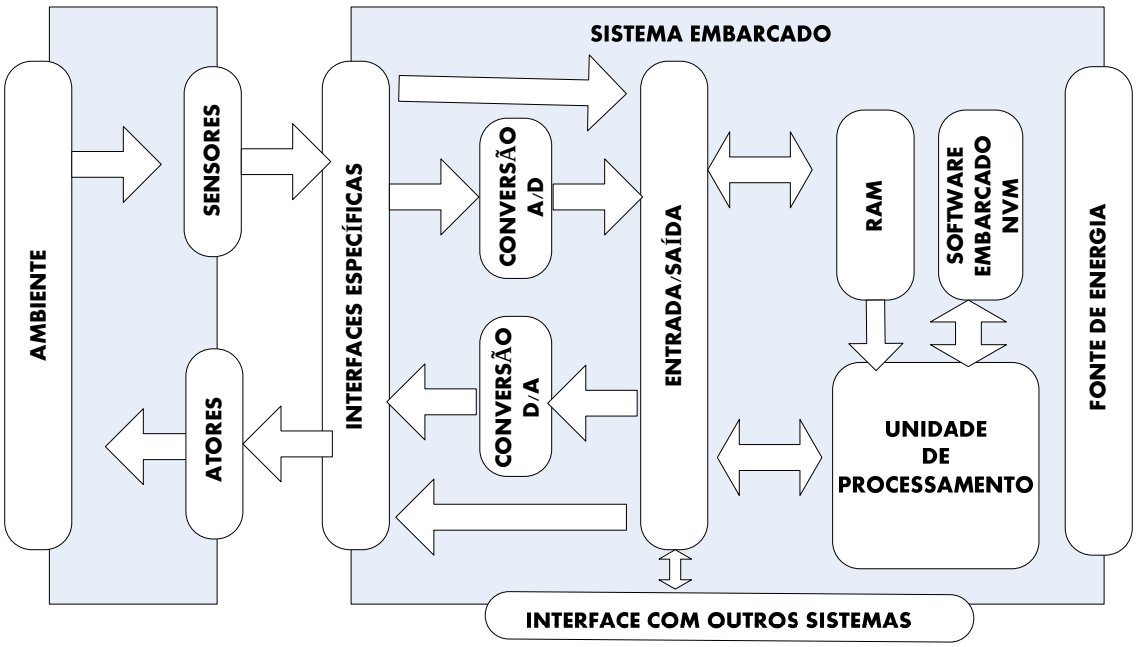

Figura 1 - Esquema genérico de um sistema embarcado (retirado de Broekman e Notenboom, 2002)

De acordo com o esquema, o software embarcado é armazenado em qualquer tipo de memória não-volátil (NVM), muitas vezes na ROM, embora ele também possa ser armazenado em cartões de memória flash, no disco rígido, etc. O software embarcado é compilado para um processador particular, a unidade de processamento, que normalmente requer certa quantidade de memória RAM para funcionar. Como a unidade de processamento só pode processar sinais digitais, enquanto o ambiente eventualmente lida com sinais analógicos, deve ser feita a conversão digital-analógico e analógico-digital. A unidade de processamento processa todas as entradas e saídas (E/S) de sinais através de uma camada de E/S dedicada. O sistema integrado interage com a planta e, possivelmente com outros sistemas (embarcados) através de interfaces específicas. Sistemas embarcados podem extrair energia a partir de uma fonte genérica ou podem ter sua própria fonte de alimentação dedicada, como baterias por exemplo (Broekman e Notenboom, 2002).

Segundo Berger (2001), os sistemas embarcados possuem as seguintes características que diferem dos sistemas tradicionais:

- São dedicados a tarefas específicas
- São apoiados por uma ampla gama de processadores e arquiteturas de processador.
- São geralmente sensíveis custo.
- Tem restrições de tempo-real.
- As implicações da falha de software, em geral, é muito mais grave
- Tem limitações de energia.
- Geralmente devem operar sob condições ambientais extremas.
- Tem muito menos recursos do sistema
- Geralmente armazenam todo o código objeto em sua ROM.
- Exigem ferramentas e métodos especializados para ser projetados de forma eficaz.

Devido à estreita associação do software embarcado com ambientes críticos e muitas vezes com riscos que ameaçam a vida, o software embarcado enfrenta exigências de qualidade elevada. Deve-se assegurar que os sistemas e seus softwares executem da forma pretendida, ou ainda melhor. Isso é relevante para toda a engenharia de software, mas é fundamental para software embarcado. Se o software embarcado tem qualidade insuficiente, prejuízos graves poderão ocorrer, com ferimentos, mortes, ou catástrofes (Ebert e Salecker, 2009).

3. Teste de Sistemas Embarcados

Conforme visto na seção anterior, os sistemas embarcados apresentam características próprias que separa o teste de sistema embarcado de um software comum. Como exemplo, pode-se citar (Berger, 2001):

- Devem executar de maneira confiável por longos períodos de tempo
- São utilizados com frequência em aplicações onde a vida humana está em risco
- São muitas vezes tão sensíveis ao custo que não há margem para ineficiências
- Devem com freqüência compensar falhas no hardware embarcado

- Eventos no mundo real são normalmente assíncronos e não determinísticos, fazendo com que testes de simulação sejam difíceis e não confiáveis

Como há vários tipos de sistemas embarcados e todos eles possuem características que podem diferir enormemente entre um e outro, uma questão que deve ser levada em conta é o sistema que está sendo testado. Por exemplo, o teste de celulares é significativamente diferente do teste de conversores de vídeo ou do sistema de controle de velocidade em automóveis. Dessa forma, não há uma abordagem única de teste para sistemas embarcados. Por outro lado, sistemas embarcados tem características únicas, ou seja, há muitos problemas semelhantes e que tem soluções semelhantes. Os princípios básicos de teste devem ser aplicados a todos os projetos de teste de sistemas embarcados, mas devem ser diferenciados, de alguma forma, para resolver seus problemas específicos (Broekman e Notenboom, 2002).

Outra questão referente ao teste de software embarcado é que ele deve ser feito sobre uma plataforma de hardware relevante. A maioria dos softwares embarcados tem um ambiente de desenvolvimento que é diferente do ambiente destino. O ambiente de desenvolvimento, muitas vezes tem simuladores e outros meios para verificar que o software funcione conforme o pretendido. Quando o software é finalizado, ele é carregado para o ambiente destino e, normalmente, ele não pode ser depurado ou alterado nesse ambiente. As mudanças necessárias são realizadas no ambiente de desenvolvimento, e uma versão compilada é novamente carregada para o ambiente de destino (Tian et al, 2009).

4. Pesquisas na área

Nessa seção são apresentadas quatro pesquisas relacionadas ao teste de sistemas embarcados com enfoque no teste estrutural. A procura por esses artigos foi feita de forma ad-hoc nas bases ACM, IEEE, Springer e Google Scholar. A seleção foi feita considerando o objetivo de investigar critérios, estado-da-arte e limitações para o teste estrutural desses tipos de sistemas .

An Industrial Case Study of Structural Testing Applied to Safety-critical Embedded Software (Guan, Offut e Ammann, 2006)

O artigo apresenta os resultados de um estudo de caso de teste de software embarcado crítico e de tempo real utilizado no setor de transportes. Nesse estudo de caso foram comparados os teste que eram realizados na própria companhia (testes funcionais realizados de forma manual - MFT) com os critérios de teste CACC (*Correlated Active Clause Coverage*), que são critérios de teste baseados em lógica.

O objetivo do CACC é testar cláusulas individuais dentro de expressões lógicas. Nesse critério, as expressões lógicas são formalizadas de maneira matemática utilizando os conceitos de predicado (que representa uma expressão que avalia um valor booleano) e cláusula (que representa um predicado que não contém nenhum operador lógico) a fim de introduzir critérios de cobertura com expressões lógicas mais simples. As expressões lógicas podem ser derivadas a partir do código-fonte, especificação ou máquinas de estados finitos. Para testar cláusulas individuais de forma a afetar o predicado, é preciso fornecer valores específicos para outras cláusulas de forma que o valor do predicado dependa diretamente da cláusula sendo testada (*major clause*).

A aplicação utilizada no estudo de caso é um sistema de controle crítico da indústria de transportes e era formada por uma coleção muito complicada de máquinas de estado e algoritmos que interagiam entre si. Além disso, essa coleção também interagia com uma quantidade grande de dispositivos de hardware, incluindo baterias, geradores, vários sensores e um comunicador wireless. Dessa sistema foram então obtidos 70 predicados sendo que, 50 desses predicados eram compostos por apenas 1 cláusula; 17 eram compostos por 2 cláusulas e 3 eram compostos por 3 cláusulas. Esses predicados resultaram em 134 testes CACC.

Como o CACC foi aplicado aos predicados sem utilizar nenhuma ferramenta, os valores foram selecionados arbitrariamente de forma manual. Devido a limitações de hardware, alguns valores das variáveis só puderam ser alcançados utilizando simulação. O CACC encontrou 3 das falhas encontradas no MFT e mais onze falhas adicionais, sendo que 1 falha poderia ter causado erro catastrófico inclusive com o risco de morte. Algumas dessas falhas só puderam ser encontradas utilizando simulação. Outro ponto importante com relação aos testes CACC nesse estudo de caso é que os predicados não tinham muitas cláusulas, sendo que é mais difícil construir testes para predicados com muitas cláusulas

Os testes funcionais foram realizados por um engenheiro da companhia (que tinha mais experiência que o testador que utilizou o CACC) de forma manual e executados diretamente no hardware sem utilizar simulação. Os resultados foram conferidos manualmente e foram encontrados 8 falhas, todas elas encontradas por meio de inspeções visuais do comportamento do hardware.

Os autores concluem dizendo que os critérios do CACC encontraram várias falhas significantes que não foram encontradas pelo MFT. Além disso, os critérios baseados em lógica podem aumentar a segurança do software industrial. Outra questão levantada por eles, é que para se testar um sistema embarcado e de tempo-real deve-se ter o conhecimento de software e hardware. Por fim, segundo os autores, o estudo trouxe evidências de que os critérios de teste estruturais podem ser úteis e compensadores para softwares embarcados e de segurança crítica.

Coverage-Based Testing on Embedded Systems (Wu et al, 2007)

O artigo descreve um estudo de caso de um teste baseado em cobertura para um software de um sistema embarcado focando na minimização do overhead durante a instrumentação.

Os autores desenvolveram um grupo de ferramentas de teste para cobertura automática – a exVantage (eXTreme Visual-Aid Novel Testing and Generation) que inclui quatro componentes principais: 1) inspetor de código automático, 2) módulo de instrumentação do programa, 3) plataforma para execução de testes automatizados e 4) um módulo de análise. Cada um desses componentes corresponde a uma ferramenta no grupo de ferramentas. A Figura 2 apresenta o *workflow* desse grupo de ferramentas.

Segundos os pesquisadores, um ponto importante no teste de cobertura, em particular para sistemas embarcados e de tempo-real, é em relação ao *overhead* de monitoramento da execução do programa. Os *probes* inseridos no programa original durante a instrumentação podem causar um *overhead* na execução do teste. Esse *overhead* pode vir de duas fontes: da exigência da CPU para execução dos *probes* ou da exigência de memória para armazenar os traces de execução. Nesse trabalho, os autores focam no

segundo problema e para isso utiliza em seu estudo de caso um sistema embarcado com arquitetura MIPS, baseado no sistema operacional Vxworks Real Time (RTOS).

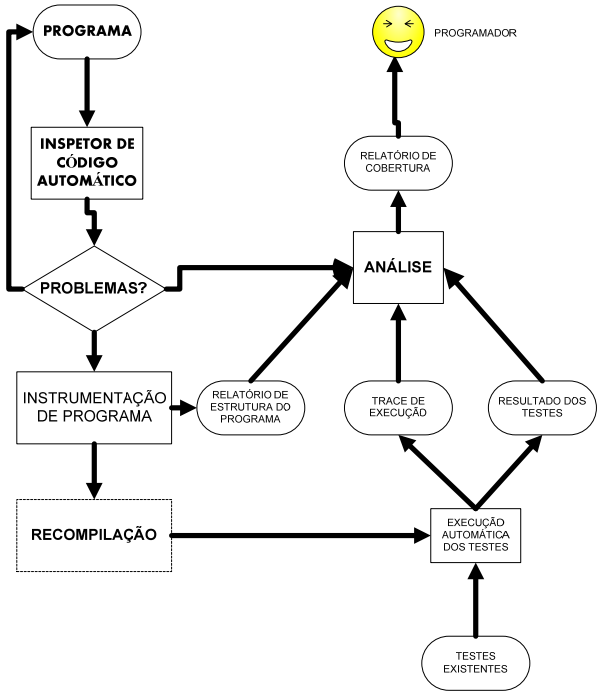

Figura 2 - Workflow do grupo de ferramentas ExVantage

Um dos problemas que eles encontraram foi que na versão original da ferramenta exVantage, o mecanismo de memória compartilhada era utilizado para manter o *buffer* de memória dos *traces*. No entanto, a imagem do VxWorks que foi utilizado no sistema do estudo de caso não tinha um componente de memória compartilhada incluído.

Para reduzir o *overhead* durante a instrumentação, a solução dos autores foi a seguinte: 1) Gravar a informação de cobertura os invés do trace de execução completo. Dessa forma, o número de *probes* e o tempo de exigência de CPU são reduzidos; 2) cada vez que um *probe* for executado, os dados são armazenados em memória em vez de serem escritos diretamente nos arquivos. Assim, o tempo de CPU para escrita em arquivos é eliminado; 3) utilizar formato de dado binário para informação de trace. Dessa forma, a exigência de memória é reduzida enormemente

Com relação à questão do compartilhamento de memória do VxWorks, os autores utilizaram a API "*sysMemtop()*" que retorna o primeiro byte de memória que não é utilizado pelo sistema nem pelo usuário. Os *traces* de memória começam a partir desse ponto para obter os blocos de memória para cada processo original.

De acordo com os experimentos feitos pelos autores, a solução tem um desempenho suficientemente bom para teste de cobertura de sistemas embarcados. Como trabalhos futuros eles pretendem aplicar a exVantage em outros sistemas, implementar um sistema de log em tempo real e desenvolver uma linha de produto.

Evolutionary white-box software test with the EvoTest Framework, a progress Report (Gross et al, 2009)

Nesse artigo, os autores avaliam a aplicabilidade do protótipo de uma ferramenta de teste estrutural evolucionário na indústria. Essa ferramenta faz parte do framework EvoTest (ETF), que também é composto por um componente de teste caixa-preta e um componente para geração de sinal. No estudo de caso, uma empresa chamada BMS (Berner & Mattner Systemtechnik GmbH) – avaliou a ferramenta em quatro módulos de software existentes.

O objetivo dos algoritmos evolucionários é executar o código sob teste com um número considerável de diferentes parâmetros de entrada a fim de maximizar a chance de encontrar erros no código. Para detectar esses erros com o mínimo de esforço, o conjunto de testes é reduzido para aquele que é suficiente cobrir a estrutura do código de acordo com a métrica de cobertura em uso.

Os algoritmos evolucionários realizam a otimização baseados no resultado de uma função fitness. O fitness da primeira geração de indivíduos é testado, e as características, conhecidas como genes, dos indivíduos mais bem adaptados são propagados para a próxima geração e o processo continua até que um dos vários critérios for atendido, por exemplo quando os indivíduos que cobriram todas as sentenças sob teste foram encontrados ou quando nenhuma sentença foi coberta e nenhuma melhora no fitness dos indivíduos ocorreu nas últimas gerações. No contexto de teste estrutural evolucionário, cada gene corresponde ao valor de uma variável de entrada específico da função em teste.

Durante a execução dos casos de teste, a função sob teste é executada de forma isolada, fora do ambiente que ela normalmente iria ser executada. Isso leva a alguns problemas, como por exemplo, os relacionados aos módulos de software que usam múltiplos contextos, como contexto de interrupção.

Para avaliar a ferramenta ETF, foram selecionados quatro estudos de caso e cada um deles consistia de um módulo de software embarcado do setor automotivo implementados em ANSI C, cujos códigos fontes vieram de duas ferramentas diferentes de geração automática de código. Os estudos de casos tiveram que ser customizados para lidar com algumas limitações da ferramenta, como por exemplo, não processar a diretiva #include e não oferecer suporte ao uso de ponteiros.

Segundo os autores, o objetivo da avaliação não foi medir o quanto a cobertura foi alcançada com a ferramenta, mas verificar se era possível gerar casos de teste para as funções contidas no estudo de caso e também se a ferramenta iria completar seu teste e apresentar um resultado. Os resultados mostraram que os casos de testes puderam ser gerados com sucesso para 37% das funções selecionadas para avaliação, que segundo os autores, foi um bom resultado para um protótipo. A razão mais comum para as falhas na geração de casos de teste foi a utilização de ponteiros.

Segundo os autores, ainda é necessário um trabalho significativo antes que essa abordagem seja aplicada na indústria. Entre os aspectos que precisam ser melhorados estão: redução de parâmetros; suporte a ponteiros; melhorar a interface gráfica da ferramenta e gerar um conjunto de casos de teste que cobrem completamente as funções que referenciam variáveis *volatile* (a utilização da palavra chave *volatile* pode ser utilizada para detectar as variáveis que podem ser modificadas fora do contexto do software atual. Durante a geração dos casos de teste a função em teste é executada isoladamente, qualquer alteração aos valores das variáveis *volatile*, que normalmente precisam ser realizadas dentro de um contexto diferente, precisa ser simulada).

Analyzing Structure-based Techniques for Test Coverage on a J2ME Software Product Line (Silva e Soares, 2009)

O artigo apresenta um estudo de caso com o objetivo de avaliar a aplicação das técnicas de teste estrutural uma linha de produto de software (LPS) Java ME, MobileMedia (MM). O foco do estudo estava nos aspectos de cobertura de teste. A ferramenta de teste JaBUTi foi utilizada a fim de fornecer informações de cobertura de teste, de acordo com as técnicas aplicadas.

As linhas de produto de software representam uma tecnologia que oferecem suporte as variações do software. Os produtos obtidos de uma linha de produto são similares mas diferem entre si. A base (núcleo) contém características obrigatórias e toda evolução ou atualização dela introduz características opcionais/alternativas resultando em um novo produto. A evolução das linhas de produto implica em mudanças de diversas naturezas, tais como: (i)introdução e remoção de características que podem ou não ser transversais, e (ii) transformação de características obrigatórias em opcionais ou alternativas e vice-versa. Essas mudanças implicam em novas definições de teste para verificar se elas causam ou não impacto em outras características existentes ou no comportamento do software em geral.

A linha de produto MM utilizada no estudo de caso é uma aplicação Java ME que manipula música, foto e vídeo para dispositivos móveis. As características do núcleo do MM são: criar/apagar mídia (música, fotografia e vídeo), rotular e exibir/iniciar mídia. As características alternativas relacionadas com tipos de mídia suportados são: música, fotografia e/ou vídeo. As características opcionais são: contagem e classificação de mídia, cópia de mídia e definição de mídias favoritas.

Foram utilizados quatro releases da LPS MM, sendo que cada release estava incluído em um cenário modificado devido a introdução de uma nova característica. A definição dos casos de teste para esse estudo foi feita com base na análise da documentação existente do MM, no comportamento de características e em conversas com os desenvolvedores. A ferramenta Jabuti foi utilizada para calcular a cobertura de teste em relação aos critérios de fluxo de dados (todos-usos-dependentes-de-exceção, todos-usos-independentes-de-exceção, todos-potenciais-usos-dependentes-de-exceção, todos-potenciais-usos-dependentes-de-exceção) e de fluxo de controle (todas-arestas-dependentes-de-exceção, todas-arestas-independentes-de-exceção, todos-nós-dependentes-de-exceção e todos-nós-dependentes-de-exceção). A porcentagem de cobertura resultante de cada critério é apresentada na Tabela 1.

Tabela 1 - Porc. de cobertura obtido na aplicação dos critérios de fluxo de dados e fluxo de controle na LPS MobileMedia

	todos-usos		**todos-pot-usos**		**todas-arestas**		**todos-nós**	
	ei	***ed***	***ei***	***ed***	***ei***	***ed***	***ei***	***Ed***
001	68	0	70	0	64	15	66	16
002	72	0	75	0	68	11	72	10
003	74	0	78	0	71	21	75	24
004	73	0	76	0	68	13	77	13

Segundo os autores, os resultados obtidos de cobertura são viáveis. Eles ainda concluem que a análise estática de código é adequada a fim de verificar os padrões de código,

analisar as sentenças e/ou a correção dos ramos ou até mesmo encontrar um código não-executável. Por outro lado, há outros fatores que fazem com que outras técnicas de teste devam ser utilizadas de forma complementar a técnicas de teste estrutural, entre elas estão as técnicas de teste funcionais e as baseadas em defeito.

Outras pesquisas na área

Há também outros esforços relacionados ao teste de sistemas embarcados que não estão ligados ao teste estrutural. Devido a diferenças entre o ambiente de desenvolvimento de um software embarcado com o ambiente destino, Tian et al (2009) desenvolveram um ambiente de teste distribuído para sistemas embarcados. Por meio desse ambiente, é possível, dentre outras coisas, gerar casos de testes, simular parte de um ambiente real, receber a saída de execução do software sob teste através do ambiente de simulação. No trabalho de Shuping e Ling (2008), os autores aprimoraram o modelo de teste V. Li et al (2009) e Arons et al (2006) abordam testes formais para sistemas embarcados.Wei-jie, Hui e Zheng-hong (2009) projetaram e implementaram uma plataforma de teste para sistemas de controle de armas de fogo. Grenning (2007) apresenta uma abordagem para aplicar o desenvolvimento guiado por testes (TDD) em sistemas embarcados.

5. Conclusão

Este artigo apresentou uma visão geral de sistemas embarcados, apresentando suas características e os desafios relacionados ao teste de tais sistemas. Além disso, foram apresentados quatro trabalhos relacionados ao teste estrutural de sistemas embarcados.

Atualmente há um grande número de sistemas embarcados e cada um deles tem características específicas que diferem um dos outros. A tendência é que esse número aumente ainda mais, aumentando também sua complexidade. Dessa forma, aliado ao fato da natureza crítica de alguns sistemas embarcados, a atividade de teste é essencial.

Com base nas pesquisas apresentadas neste artigo, nota-se que o teste estrutural, em sua essência, não é diferente para sistemas embarcados. Além disso, não foram encontradas propostas de novos critérios.

Outro ponto observado é que as pesquisas relacionadas ao teste estrutural de sistemas embarcados, em sua maioria, tratam de estudos de caso para uma aplicação específica em que se tenta verificar como realizar o teste dadas as restrições dos sistemas em estudo. Entre essas restrições, podem-se citar limitações de memória, CPU, desempenho, etc. Outro ponto é garantir se o teste não afeta o sistema em si, como por exemplo, no trabalho de Wu et al (2007) em que a instrumentação poderia causar *overhead* de memória.

Outra questão que deva ser considerada é a simulação. No trabalho de Guan, Offutt e Ammann (2006), os autores citam que alguns defeitos só puderam ser descobertos utilizando simulação. Gross et al (2009) reforça essa idéia ao comentar que as alterações nos valores de variáveis que podem ser modificadas fora do contexto do software precisam ser simuladas.

Referências

Guan, J., Offutt, J., e Ammann, P. (2006). “An industrial case study of structural testing applied to safety-critical embedded software”. In Proceedings of the 2006 ACM/IEEE international Symposium on Empirical Software Engineering (Rio de

Janeiro, Brazil, September 21 - 22, 2006). ISESE '06. ACM, New York, NY, 272-277

Wu, X., Li, J. J., Weiss, D., and Lee, Y. (2007). “Coverage-Based Testing on Embedded Systems”. In Proceedings of the Second international Workshop on Automation of Software Test (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 7.

Gross, H., Kruse, P. M., Wegener, J.;Vos, T. (2009). “Evolutionary White-Box Software Test with the EvoTest Framework: A Progress Report”. In *Proceedings of the IEEE international Conference on Software Testing, Verification, and Validation Workshops* (April 01 - 04, 2009). ICSTW. IEEE Computer Society, Washington, DC, 111-120.

Silva, L. Soares, S. (2009). “Analyzing Structure - based Techniques for Test Coverage on a J2ME Software Product Line”. In Proceedings of the 10th Latin-American Test Workshop (March 02 – 05, 2009). Buzios, Rio de Janeiro, Brasil.

Ebert, C.; Salecker, J.(2009) , "Guest Editors' Introduction: Embedded Software Technologies and Trends," *Software, IEEE* , vol.26, no.3, pp.14-18, May-June 2009

Berger, A. (2001) "Embedded Systems Design: An Introduction to Processes, Tools and Techniques", CMP Books

Broekman, B; Notenboom, E.(2002)”Testing Embedded Software”. Addison-Wesley Professional

Tian, P.; Wang, J; Leng, H; Qiang, K (2009) "Construction of Distributed Embedded Software Testing Environment," International Conference on Intelligent Human-Machine Systems and Cybernetics, 2009, vol. 1, pp.470-473

Grenning, J.; (2007), "Applying test driven development to embedded software," *Instrumentation & Measurement Magazine, IEEE* , vol.10, no.6, pp.20-25, December 2007.

Shuping, L; Ling, P; (2008) "The Research of V Model in Testing Embedded Software," *Computer Science and Information Technology, 2008. ICCSIT '08. International Conference on* , vol., no., pp.463-466, Aug. 29 2008-Sept. 2 2008

Wei-jie, H.; Hui, Y.; Zheng-hong, D.; (2009) "Design and Implementation of the Embedded Software Testing Platform for the Gun Fire Control System," *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on* , vol., no., pp.1-4, 11-13 Dec. 2009

Grenning, J.; (2007) "Applying test driven development to embedded software," *Instrumentation & Measurement Magazine, IEEE* , vol.10, no.6, pp.20-25, December 2007

Li, Z.; Liu, B.; Ma, N.; Yin, Y.(2009) "Formal testing applied in embedded software," *Reliability, Maintainability and Safety, 2009. ICRMS 2009. 8th International Conference on* , vol., no., pp.697-702, 20-24 July 2009

Arons, T.; Elster, E.; Murphy, T.; Singerman, E. (2006) "Embedded Software Validation: Applying Formal Techniques for Coverage and Test Generation," *Microprocessor Test and Verification, 2006. MTV '06. Seventh International Workshop on* , vol., no., pp.45-51, 4-5 Dec. 2006