Uma Proposta de Utilização de uma Arquitetura Cognitiva no Processo de Teste de Software

Virgínia Mara Cardoso

Departamento de Engenharia de Computação e Automação Industrial - Faculdade de Engenharia Elétrica e de Computação - Universidade Estadual de Campinas
13.083-970 – Campinas – SP – Brazil

vcardoso@dca.fee.unicamp.br

Resumo. *Muitas são as questões sobre o comportamento cognitivo, estudado em várias disciplinas. Um desafio de trabalhar com a teoria cognitiva na área computacional é a tentativa de encontrar um conjunto de mecanismos e estruturas computacionais praticáveis que possam responder questões sobre o comportamento cognitivo. A partir desta idéia desenvolveram arquiteturas cognitivas onde mecanismos e estruturas fixas sustentam a cognição humana. A atividade de teste de software exige conhecimentos específicos em determinadas etapas o que torna o processo complexo não podendo ser totalmente automatizado. Neste artigo são feitas algumas propostas para a utilização de um sistema cognitivo em etapas complexas do processo de teste que exigem um indivíduo especializado para tomar decisões relevantes, podendo assim esta atividade despender menos tempo, custo e pessoas.*

1. Introdução

A sistemática do processo de teste de software exige um planejamento, projetos de casos de teste, execução do teste e avaliação dos resultados. Para auxiliar determinadas fases complexas existem ferramentas automáticas. Mas o processo de teste para completar-se necessita de um testador. É exigido deste testador o conhecimento detalhado de todas as etapas desta atividade e também o programa a ser testado. Neste trabalho propomos a execução de testes de software genéricos com o auxílio da ferramenta POKETOOL [Chaim, 1991] em algumas etapas. Nas fases que exigem cognição é proposta a utilização de um sistema cognitivo, a arquitetura SOAR [Laird e*t. al*, 1986], podendo assim automatizar mais uma parte do processo de teste. Com esta nova abordagem é possível, através de um histórico testar pontos mais suscetíveis a erros e aplicar critérios mais específicos.

No processo de teste o testador deve participar de algumas etapas como a elaboração dos casos de teste, que são dados de entrada e resultados esperado, a análise dos resultados finais e a verificação da continuidade dos testes. As outras etapas podem ser automatizadas e assim não exigem esforço algum da equipe envolvida. Para aumentar a confiabilidade do processo, redução de tempo e custo, as etapas que envolvem o testador merecem uma atenção especial. Toda a especialidade e conhecimento deste indivíduo são fatores relevantes no resultado final, sobrepondo ao uso e escolha da ferramenta.

Na área de teste muitas ferramentas já foram desenvolvidas para auxiliar testes genéricos e também para casos específicos, mais atreladas a linguagem e não as especificações da estrutura do programa. Dentre elas podemos citar a POKETOOL [Chaim, 1991], a PROTESTE [Price e Zorzo, 1990], a Test Inc [Lutz, 1990] e a PHYTHIA [Vokolos, *et. al* 1997].

A respeito de sistemas cognitivos são citadas várias arquiteturas com inúmeras vantagens, como algumas que possuem um framework concreto onde é possível uma modelagem detalhada e específica do fenômeno cognitivo. Dentre elas podemos citar: SOAR [Laird *et. al*, 1986], ACT-R [Anderson, 1993], CLARION [Sun, 2002], dentre outras. Quando referimos ao assunto que diz respeito a um teste de software cognitivo ou um processo de teste que utiliza sistemas cognitivos, testes que utilizam ou são feitos em arquiteturas cognitivas, até o presente momento não são encontrados artigos ou relatos que tratam do assunto.

Neste contexto foi escolhida a arquitetura cognitiva SOAR, uma arquitetura cognitiva genérica que utiliza conhecimentos e

métodos para solucionar problemas que requerem capacidade cognitiva. A ferramenta de teste POKETOOL foi selecionada para fazer parte da atividade de teste em algumas fases, uma vez que também é utilizada para auxiliar no processo de teste de programas genéricos, escritos na linguagem C.

O artigo está organizado da seguinte forma. A próxima seção apresenta o processo de teste de software e a ferramenta POKETOOL. A seção 3 descreve a arquitetura cognitiva SOAR. As propostas para a utilização de um sistema cognitivo no processo de teste estão na seção 4. Finalmente, a seção 5 conclui o artigo.

2. O Teste de software e a ferramenta POKETOOL

Para testar o software, basicamente utilizam-se técnicas funcionais e estruturais. A técnica funcional, também chamada de teste caixa-preta, não considera os detalhes da implementação, baseia-se na especificação para derivar os requisitos de teste. Já a técnica estrutural baseia-se em informações da implementação, do texto do programa, e recebe o nome de teste caixa-branca.

A condução sistemática do teste requer planejamento, projetos de casos de teste, execução e avaliação dos resultados. São denominados casos de teste os conjuntos que têm dados de entrada para o programa em teste e também a saída esperada para esse dado. O projeto de caso de teste deve ser muito bem elaborado para que tenha a maior probabilidade de encontrar defeitos, com o mínimo de tempo e esforço possíveis. Através da execução de casos de teste os resultados são comparados, ou seja, o resultado obtido e o resultado esperado, mas é necessário um mecanismo ou mesmo uma pessoa, para identificar se houve ou não a ocorrência de falhas através de um oráculo. Este oráculo não vai encontrar e corrigir os defeitos, este seria um processo de depuração, mas sim dizer se os resultados encontrados estão corretos ou não.

Os requisitos estabelecidos pelas técnicas de teste estabelecem os critérios de teste com a função de auxiliar o testador selecionar os dados de teste e decidir se um produto foi suficientemente testado. É um critério de teste que serve para direcionar a atividade de teste, ou seja, dita as condições a serem utilizadas no teste. Como exemplo de critérios de teste estrutural baseados em fluxo de controle podemos citar todos-os–nós, todos-os-arcos, todos-os-caminhos que requerem o exercício de cada nó, cada arco e caminho respectivamente. Mas a técnica de teste estrutural possui também critérios baseados em fluxo de dados e outros.

Em decorrência do crescimento do tamanho e da complexidade de sistemas de software tornou-se necessário o uso de ferramentas automáticas para auxiliar no teste e na análise dos resultados do teste. Existem muitas ferramentas automatizadas que auxiliam todo ou parte do processo de teste. Por exemplo, a ferramenta POKE-TOOL (Potencial Uses Criteria Tool for Program Testing) foi desenvolvida por Chaim [1991] para dar suporte ao teste estrutural de programas baseado em análise de fluxo de dados. A ferramenta apóia a aplicação de critérios de teste estruturais. Trata-se de uma ferramenta de teste de unidade de programas, multi-linguagem, ou seja, não está atrelada a uma linguagem de programação específica. É uma ferramenta interativa, que disponibiliza ao usuário todas as tarefas básicas de um teste: a análise do código fonte e a criação de uma base de dados; a geração de relatórios baseados na análise estática do código fonte; a instrumentação do código fonte; a análise dos resultados, gerando relatórios; e a geração de relatórios para auxiliar a organização das atividades de teste e a derivação de conjunto de casos de teste específicos.

A sessão de trabalho da POKE-TOOL está dividida em duas partes: a fase estática e a fase dinâmica. Na primeira fase é feita uma análise estática da unidade em teste, o programa em teste é instrumentado, gerando uma nova versão para o teste propriamente dito. A partir de informações do código fonte, obtém-se o grafo de fluxo de controle e o grafo de fluxo de dados, sendo então gerados os elementos requeridos para os critérios determinados. A fase dinâmica consiste no processo de execução e avaliação dos casos de teste aplicados. É executado o programa instrumentado e a ferramenta armazena em arquivos as informações obtidas em tempo de execução tais como: os dados de entrada, os parâmetros fornecidos, as saídas obtidas e os caminhos executados para cada caso de teste. Após a execução da unidade em teste, o módulo avaliador analisa as informações recebidas. Este módulo determina a cobertura atingida com o conjunto de casos de teste aplicados para o critério escolhido e fornece os elementos requeridos executados e os não executados.

Na próxima seção será descrito o sistema cognitivo utilizado para auxiliar no

processo de teste nas etapas que exigem um testador.

3. A Arquitetura SOAR

Toda a descrição da arquitetura SOAR foi descrita segundo [Lehman *et. al*, 2006; Gorski e Laird, 2006; Soar, 2006; Young e Lewis, 1998].

Uma arquitetura cognitiva é abrangente, possui estruturas e processos essenciais de um modelo cognitivo computacional de domínio genérico, utilizado em uma análise ampla, em múltiplos níveis e múltiplos domínios, dos fenômenos da cognição e do comportamento [Newell, 1990; Sun, 2002]. Pode-se dizer que se trata de um conjunto fixo de mecanismos e estruturas que processam dados para produzir comportamento e ao mesmo tempo, do ponto de vista teórico, mostra pontos comuns de comportamentos cognitivos. A SOAR (State, Operator and Result) foi desenvolvida por John Laird, Allen Newell e Paul Rosenbloom em 1987 na Carnegie-Mellon University. Esta arquitetura foi projetada para ser um sistema que pudesse suportar múltiplas resoluções de métodos de problemas para muitos e diferentes problemas.

A SOAR é projetada para ser uma arquitetura cognitiva genérica, capaz de usar uma larga escala de métodos e de conhecimento para resolver os problemas que requerem a integração de inteligência geral subjacente de muitas potencialidades cognitivas. Sua estrutura produz um meio para organizar o conhecimento como uma seqüência de decisões através de um espaço de problema. Esta representação permite descrever todas as possíveis ações de um ponto de vista estático e também dinâmico, mostrando o possível comportamento.

Para descrever o problema de estados a arquitetura suporta estruturas básicas que são os estados e operadores. Os estados contêm toda a informação sobre a situação corrente, incluindo percepção e a descrição de metas correntes e espaços de problemas. Os operadores são responsáveis por ocasionar passos no espaço de problemas. A mudança de um estado para o outro ocorre através da aplicação de um operador no estado corrente. Para capturar a dicotomia entre o conhecimento genérico e uma aplicação específica a SOAR possui dois diferentes tipos de memória, como ilustrado na Figura 1[Lehman *et. al*, 2006]. O primeiro inclui uma memória declarativa interna que mantém informações específicas para realizar tarefas da situação atual, incluindo o controle de informações. Esta memória é conhecida nas arquiteturas de sistemas de produção como memória de trabalho (*working memory* - WM). Esta memória contém percepções e hierarquia de estados e seus operadores associados. O seu conteúdo pode acionar a memória de longo prazo ou ações motoras. A outra memória é referida como memória dinâmica ou Memória de Longo Prazo (*long-term memory*- LTM). Esta segunda parte trata-se do mecanismo de aprendizagem através do qual adquire novas regras de produção, onde é possível produzir comportamento. Este processo de produzir novas regras é denominado "chunking", e as novas regras recebem o nome de "chunks".

A memória LTM está dividida em três módulos. O primeiro a memória procedural que possui regras e é responsável por controlar o comportamento e mapear diretamente os operadores de conhecimento. A memória semântica que possui estruturas declarativas e a memória episódica que codifica conhecimento como episódios. A memória procedural é acessada automaticamente durante o ciclo de decisão enquanto as memórias semântica e episódica são acessadas deliberadamente através da criação de dicas específicas na memória de trabalho. As memórias de longo prazo são impenetráveis, não podem ser examinadas diretamente, certos procedimentos recuperam informações nas memórias de longo prazo e armazenam na memória de trabalho.

A SOAR possui uma interface perceptiva/motora que permite o mapeamento do mundo externo para as representações internas na memória de trabalho e de representações internas para o mundo externo. Toda percepção e ação podem acontecer em paralelo com o processo de cognição.

Para suportar cognição a SOAR possui um processo arquitetural básico denominado ciclo de decisão, onde são selecionados e aplicados os operadores. Este ciclo é composto de três fases: elaboração, decisão e aplicação. A fase de elaboração envolve o acesso paralelo à memória de longo prazo para elaborar o estado, sugerir novos operadores e avaliar os operadores. A ausência de ação define o momento de encerrar a fase de elaboração e iniciar a fase de decisão. Durante esta fase o procedimento de decisão interpreta a linguagem de preferência por operadores. O resultado desta fase é uma mudança para selecionar um operador ou um impasse de preferências incompletas ou conflito. Seguindo a decisão é a fase de aplicação, onde as regras disparam para modificar o estado. A seleção de um único operador por ciclo de decisão impõe um gargalo

cognitivo a arquitetura, isto é, um limite no trabalho cognitivo por ciclo. Os impasses acontecem automaticamente, o conhecimento elicitado pelo estado corrente não é suficiente para o procedimento de decisão selecionar um operador. Estes sinalizam uma falta de conhecimento e ocorre uma oportunidade para aprendizagem. A linguagem de impasse é definida independentemente de qualquer domínio. Quando ocorre um impasse, a arquitetura automaticamente inicia a criação de um novo sub-estado cuja meta é resolver o impasse. Deste modo, o impasse impõe uma hierarquia de metas/sub-estados no contexto da memória de trabalho.

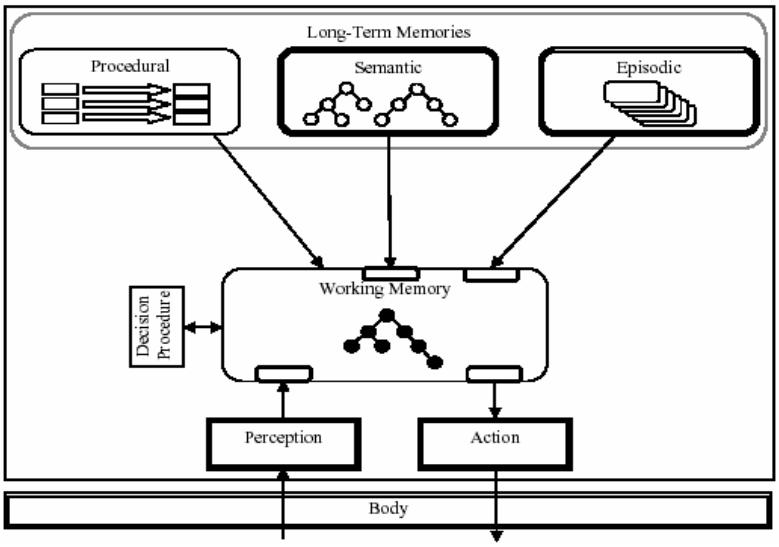

Figura 1: Estruturas da memória em SOAR.

A SOAR possui quatro mecanismos de aprendizagem. O chunking, que cria automaticamente novas regras na memória de longo–prazo utilizando-se dos resultados gerados de um impasse e previnem que um impasse ocorra em situações similares no futuro. Aprendizagem por reforço que ajusta valores das preferências por operadores; aprendizagem episódica que armazena a história das experiências e aprendizagem semântica que captura asserções declarativas mais abstratas.

Esta arquitetura trabalha com regras na estrutura de "IF" e "THEN", sendo o "IF" a condição da regra e "THEN" a ação da regra. Se as condições da regra são verdadeiras na situação atual, a ação é executada e envolve mudanças na WM. Sendo a condição verdadeira uma regra é disparada e pode provocar o disparo de outras. Toda a WM é organizada como uma estrutura de grafos de estados.

São várias as questões sobre esta arquitetura e seu funcionamento. Como as aplicações são diversas, a arquitetura por si só não responde várias questões sobre comportamento cognitivo, se de fato suporta toda cognição. No entanto esta arquitetura é uma ferramenta que proporciona a oportunidade de combinar dados teóricos e ver como eles interagem.

4. O Teste Cognitivo

Para melhor descrever a proposta será apresentado um teste estrutural sistemático com todas as suas fases, como ilustrado na Figura 2. Para este processo de teste de um programa genérico é utilizada a ferramenta POKETOOL e necessita-se também um testador ou uma equipe de especialistas. Acompanhando a Figura 2, o processo de teste inicia-se fornecendo a POKETOOL o programa que será testado e escolhendo os critérios que serão aplicados. Nesta primeira fase, teremos como produto o fluxo de controle do programa em teste, o fluxo de dados, o programa instrumentado e os elementos requeridos para o teste. Os elementos requeridos fornecidos são nós, caminhos ou associações que devem ser exercitados de acordo com o critério selecionado. A próxima fase é a elaboração dos casos de teste a partir

dos elementos requeridos obtidos. Para o teste estrutural de programas convencionais um caso de teste é o conjunto formado por dados de entrada e o resultado esperado para esses dados entrada. Os dados de entrada para unidades de programas convencionais são valores atribuídos às variáveis de entrada, ou seja, variáveis locais e globais.

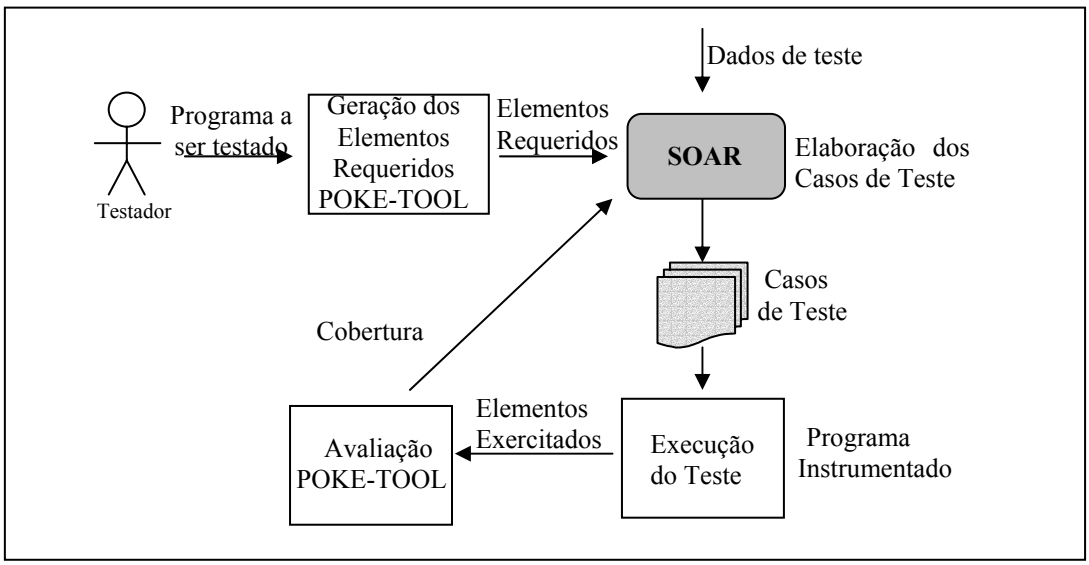

Figura 2: Processo de teste utilizando a SOAR.

A grande tarefa do testador é elaborar casos de teste para o teste propriamente dito. Trata-se de uma fase que exige muita habilidade, podemos dizer que nesta etapa necessita-se de cognição. Estes conjuntos de casos de teste bem formados podem determinar um teste satisfatório com uma percentagem grande de corretude, sabendo-se que foram exercitados pontos relevantes do programa em teste. Neste caso diminui o tempo do processo de teste, não sendo necessário voltar a fase inicial, podendo concluir o papel fundamental do testador para propiciar um processo de teste satisfatório. Em geral, nesta etapa são formadas equipes de testadores para que diferentes raciocínios possam influenciar em atacar pontos obscuros e respeitáveis do programa. Com a arquitetura SOAR podemos modificar esta fase não despendendo tanto tempo do testador ou de sua equipe. A partir de somente um conjunto de dados de teste fornecido pelo testador a SOAR geraria novos dados baseando-se nas proximidades dos valores dados. Neste caso teríamos como resultado vários conjuntos de dados de teste. Uma segunda sugestão seria toda a elaboração de casos de teste executada pela SOAR. Neste outro procedimento seria imprescindível uma modelagem específica do programa em teste, esta modelagem visa indicar nós que possuam predicados, e nós dependentes ditados pelo uso e definição de variáveis. Para a SOAR cada nó do programa em teste passaria a ser uma regra onde os seriam ditos os possíveis valores. Os nós dependentes seriam regras disparadas na seqüência da dependência de valores, podendo assim obter um conjunto de dados de teste válidos para o exercício de nós, associações e caminhos requeridos. Nesta geração de dados de teste observa-se muitos valores indevidos e indiferentes que são atribuídos as variáveis. Estes valores passariam a fazer parte de um histórico caso a necessidade de nova elaboração de casos de teste já teria um conjunto a ser descartado para aquele caso. Mas para o caso de teste é necessário também os resultados esperados e estes só podem ser determinados com os cálculos e procedimentos que o programa em teste deve executar. Para isto é especificada uma regra que dita o objetivo do programa em teste com todos os cálculos. Para cada conjunto de dado de teste fornecido pela SOAR também é calculado o resultado esperado com esta outra regra. Assim completa-se a fase que era necessário cognição e agora com um sistema cognitivo e não com um indivíduo especializado. Com os casos de teste, o teste propriamente dito é executado, ou seja, o

programa é exercitado com os valores dados e como resultado temos os caminhos exercitados. Este produto é avaliado pela POKETOOL e temos como resultado os elementos requeridos exercitados, os elementos requeridos não exercitados e a cobertura atingida de acordo com os critérios selecionados. A cobertura é a percentagem obtida confrontando elementos requeridos exercitados e não exercitados. Nesta etapa é necessário observar os valores de cobertura atingido em cada caso de teste. Necessita-se do testador e de sua equipe para verificar se o percentual de cobertura está bom e se o teste está completo ou se novos casos de teste devem ser elaborados. Se o testador julgar que o valor de cobertura não é ideal o processo de teste iniciará o seu ciclo novamente, até que estes valores estejam ideais, ou seja, com um valor tendendo a 100%. Com os valores de cobertura atingidos pelos conjuntos de dados de teste pode-se novamente utilizar a SOAR. Tratando cada conjunto de dados de teste como uma regra e seus elementos com seus respectivos valores de cobertura, a SOAR, nesta fase deve combinar os dados de dados de todos os conjuntos com seus respectivos valores de cobertura, tentando nesta combinação maximizar o valor de cobertura, melhorar o valor de cobertura sem precisar executar o programa e sem a necessidade da avaliação da POKETOOL.

5. Conclusão

Este trabalho apresentou propostas de pesquisa para a utilização da arquitetura cognitiva SOAR na atividade de teste. Como é um passo inicial foram apenas abordados os pontos onde há necessidade de um processamento inteligente com tomadas de decisão que até nos dias atuais são feitas por equipes de especialistas. Foi mostrada a necessidade de um sistema cognitivo para sinalizar a continuidade das etapas de teste durante o processo. Contudo, trabalhos futuros serão necessários para aprofundar a pesquisa, validar, estender e assim poder automatizar toda a atividade de teste de software.

6. Referências

Anderson, J. R., 1983. "*The Architecture of Cognition"*. Cambridge, MA: Harvard University Press.

Chaim, M.L., 1991. "*POKE-TOOL – Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados*", dissertação de mestrado, Faculdade de Engenharia Elétrica, Unicamp, Campinas, SP, Brasil.

Gorski, N.A., e Laird, J.E., 2006. "*Experiments in Transfer Across Multiple Learning Mechanisms*". Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning. Pittsburgh, PA.

Laird, J.E., Rosenbloom, P.S., Newell, A. 1986. "*Chunking in Soar: The Anatomy of a General Learning Mechanism.*". MachineLearning, 1, 11-46.

Lehman, J.F., Laird, J.,Rosenbloom, P., 2006. *"A Gentle Introduction to Soar, an Architecture for Human Cognition: 2006 Update"*. It updates the original paper, Sternberg, S. & Scarborough, D. 1996. (Eds), "*Invitation to Cognitive Science"*, Volume 4. Cambridge, MA: MIT Press.

Lutz, M., 1990. "*Testing Tool*", IEEE Software,Vol. 7, No. 3, Maio 1990, pp 57.

Price, M., e Zorzo, A., 1990. "*Vizualizando o Fluxo de Controle de Programas*", in Proc. IV Simp. Bras. De Eng. De Software, Águas de São Pedro, S.P.

Soar, 2006. http://sitemaker.umich.edu/soar, visitada em 29 de junho de 2006.

Sun, R., 2002. "*Duality of Mind",*Mahwah, NJ: Erlbaum.

Vokolos, F.I., Hill, M., e Frankl, P.G., 1997. "*Pythia: A regression test selection tool based on textual differencing*". Polytechnic University, Brooklyn, NY, USA.

Young, R.M. and Lewis, R.L., 1998. "*The Soar Cognitive Architecture and Human Working Memory*"**.** In Miyake, A. and Shah, P. (Eds), Models of Working Memory: Mechanisms of Active Maintenance and Executive Control. New York: Cambridge University Press.