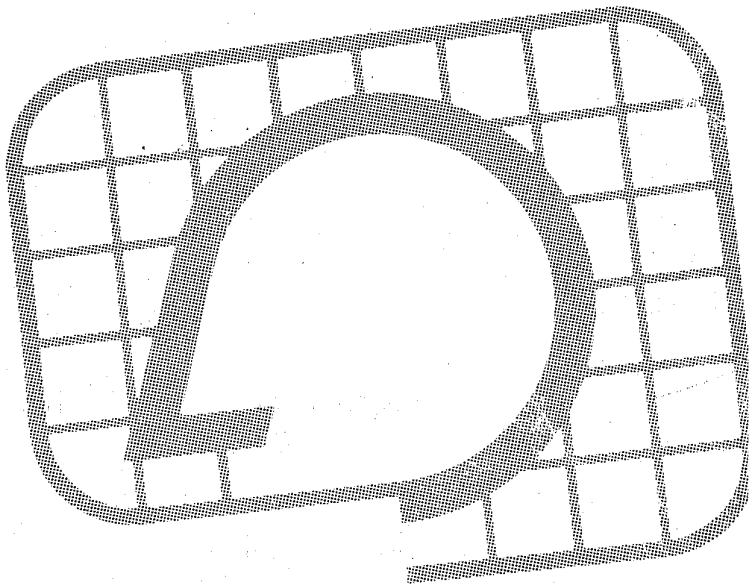
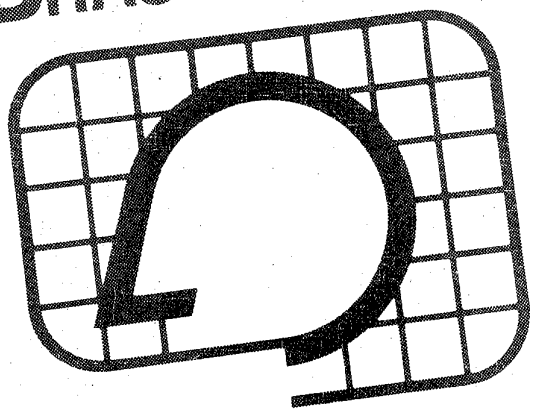


2º Simpósio Brasileiro de Inteligência Artificial



ANAIS

006.306
S612

2º SIMPÓSIO BRASILEIRO DE INTELIGÊNCIA ARTIFICIAL

20-21-22 DE NOVEMBRO DE 1985

INPE - INSTITUTO DE PESQUISAS ESPACIAIS

SÃO JOSÉ DOS CAMPOS - SP

BASES DE CONHECIMENTO

SOBRE MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE

Carlos José Pereira de Lucena

Jeferson Ferreira Soares

Departamento de Informática

Pontifícia Universidade Católica

Rio de Janeiro

1. Considerações Gerais

Existe, no âmbito da comunidade técnico-científica que atua na área de engenharia de software, um consenso sobre a necessidade de um melhor entendimento do processo de desenvolvimento de sistemase programas. Os métodos e técnicas existentes para o projeto e produção de software aceitam, como referência básica, o conhecido modelo de ciclo de vida de produtos software (1).

A presente comunicação discute alguns problemas centrais relacionados à passagem do atual paradigma para a construção de software, que é essencialmente baseado em pessoal (produção artesanal), para um paradigma formalizável e, portanto, passível de ser instrumentado por métodos computacionais eficientes (assistido por computador). Atualmente, as tecnologias disponíveis para acompanhamento do desenvolvimento de um software (ferramentas para apoio aos métodos existentes) são ainda bastante precárias. Parte do problema é a necessidade de um melhor entendimento dos métodos e metodologias existentes, que em geral, seguem a informalidade do atual modelo de ciclo de vida. Este entendimento se faz necessário como um primeiro estágio para a proposta de novos e melhores métodos e para esse fim estamos desenvolvendo técnicas para a aquisição de conhecimento sobre os aspectos formais e informais (heurísticos) de métodos de desenvolvimento de software. A inteligência artificial se ocupa da mecanização de tarefas complexas e intensivas em conhecimento: certamente a modelagem de métodos de desenvolvimento de software é uma aplicação promissora para estas técnicas.

2. Um Modelo Tentativo para o Processo de Desenvolvimento de Software

O modelo esboçado nesta seção tem em vista um processo de desenvolvimento de software que possa ser amplamente assistido por sistemas automatizados. As principais dificuldades já apontadas a respeito do modelo existente são as seguintes: as especificações utilizadas para as diversas fases do processo de desenvolvimento (requisitos, projeto e codificação) são informais; o uso de protótipos é incomum; protótipos (quando existem) são produzidos manualmente; a intenção do usuário é confrontada com o código final produzido pelo desenvolvimento; protótipos são descartados; a codificação (implementação) é manual; os testes são aplicados ao código de programas; a manutenção é exercida sobre o código fonte da implementação; as decisões sobre projeto são perdidas; a manutenção é processada através da introdução de "remendos" na documentação associada à diversas fases do processo de desenvolvimento.

Um novo paradigma, capaz de superar diversas das dificuldades que acabam de ser enumeradas, pode ser formulado da maneira indicada na figura 1, que combina propostas extraídas das referências (2) e (3).

Temos presente que um modelo como o proposto acima requer, para a sua realização, a superação de diversas dificuldades relacionadas com várias fases do processo de desenvolvimento. Até hoje não existe uniformidade entre os formalismos de especificação utilizados para o nível de requisitos, projeto (que se elimina no novo modelo proposto) e programação. Na maior parte das vezes, requisitos vêm sendo expressos através de modelos gráficos de computação (ex: Petri nets, R.nets etc.), projetos ("designs"), através de mecanismos para "programação em ponto grande" (4) (ex: Linguagens para interconexão de módulos expressos como "templates", PDL's etc) e programas através de sentenças do cálculo de predicados, apresentações algébricas, pré-condições mais fracas, funções recursivas etc.

A atenção recente que a programação em lógica vem merecendo (não confundir com a polêmica sobre a con

veniência do uso da linguagem PROLOG para codificação de aplicações da inteligência artificial), está muito relacionada ao papel reservado para uma notação para especificação/programação no estudo da revisão do modelo clássico do processo de desenvolvimento de software. Somam-se à força da notação (lógica formal) a possibilidade de associação a ele de várias estratégias heurísticas originadas de estudo de métodos de solução de problemas em inteligência artificial, que, segundo Rich (5) é a disciplina que estuda a solução de problemas considerados difíceis num certo estágio de desenvolvimento da ciência da computação. Tem-se especulado sobre a aplicabilidade da programação em lógica a análise formal (executável) na área de software (6) e (7) e a definição de linguagens interconexão de módulos (8). No nível de projetos de linguagem de programação, buscam-se modelos melhores do que o PROLOG para a implementação do conceito de programação em lógica (ex: APES (9) e SAFO (10)) e o aperfeiçoamento de técnicas da área de transformação de programas que permitam a concretização do lado direito do diagrama da figura 1. Neste contexto, entende-se uma seqüência de transformações como um programa executável para a implementação de uma especificação.

3. A Formalização através da Programação em lógica de Métodos de Desenvolvimento de Software.

Existem dois problemas centrais que precisam ser resolvidos para que o modelo apresentado na figura 1 possa vir a ter aplicabilidade. É preciso entender os fundamentos dos métodos e metodologias que diferentes especialistas adotam ao efetuar uma análise de requisitos e a derivação de um programa. Isto implica sobretudo em entender a heurística correspondente que possa permitir o fechamento de dois "loops" apresentados nas duas fases da figura 1.

Tratamos, em nossos trabalhos, de formalizar todo o conhecimento disponível sobre a análise estruturada (11), sobre o uso de lógica em análise de sistemas (7), sobre análise baseada na estruturação de dados (12 e 13) e sobre metodologias como a de Lehman (14), para citar a-

penas alguns trabalhos em andamento. Expressá-los na forma de programas em lógica nos tem permitido: a) desenvolver sistemas especialistas sobre os métodos, ou seja, representar todo o conhecimento estruturado ou não (conhecimento heurístico) que os métodos encerram sobre a análise e formulação de problemas de software; b) produzir ferramentas experimentais para os métodos, na medida em que especificações em programação em lógica são executáveis (são uma forma de implementação).

Num estágio mais avançado, estaremos identificando as estratégias que melhor modelam métodos (usando, por exemplo, SAFO (10) como linguagem de programação em lógica) e analisando as bases de conhecimento acumuladas sobre métodos para definir com maior rigor seus ambientes de desenvolvimento (ferramentas de software de suporte).

Para ilustrar os conceitos expressos acima, apresentamos a seguir uma das dez regras de execução em PROLOG previstas em (11) para execução simbólica de Diagramas de Fluxo de Dados (metade das regras valem para o nível geral do diagrama e metade para os níveis definição mais internos).

O elemento x=s do diagrama à esquerda é executável porque ele é reconhecido como um elemento externo, de destino, que recebe ativações válidas de um sub-sistema Y também válido. As regras à direita expressam esta especificação em PROLOG (figura 2).

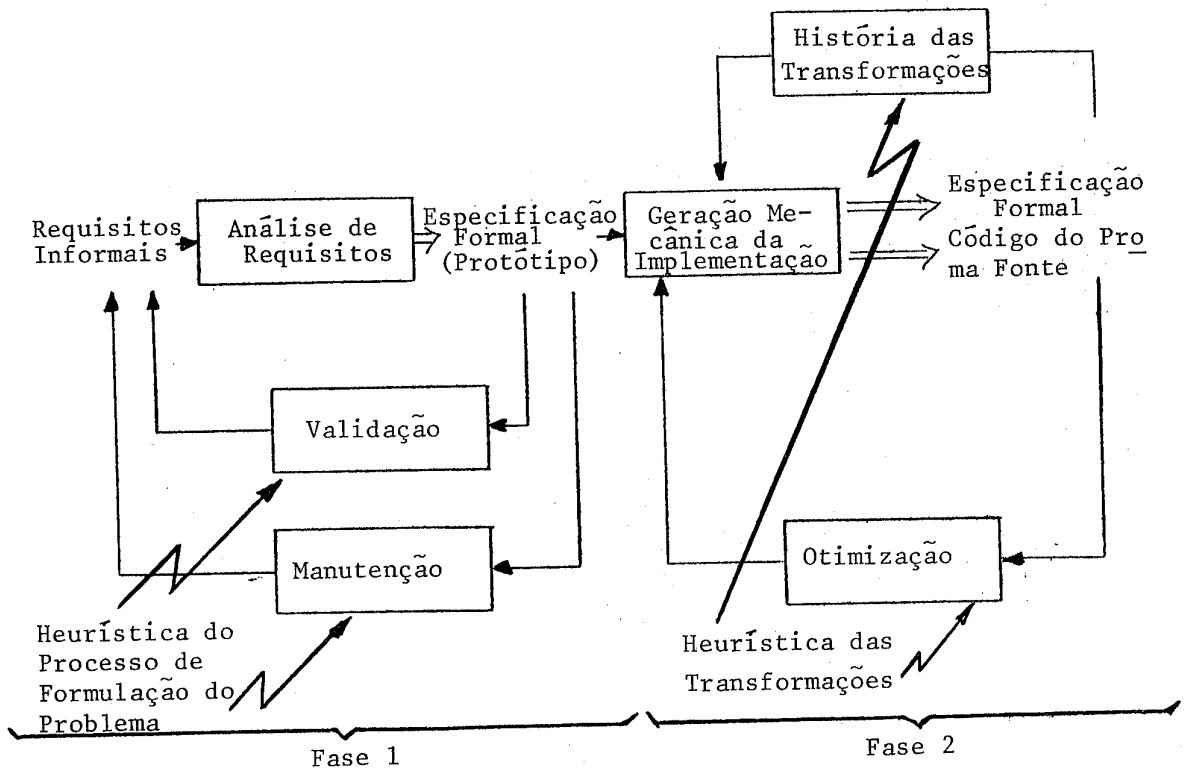
4. Ambientes de Desenvolvimento Centrados em Métodos de Desenvolvimento de Software.

Os estudos teóricos descritos nas seções anteriores têm, em última análise, objetivo de produzir tecnologias mais sólidas para a área de engenharia de software. Hoje, a engenharia de software baseia-se em ferramentas isoladas que instrumentam conceitos mais ou menos vagos, associados ao modelo clássico do processo de desenvolvimento de software. A evolução que percebemos na área de projeto de software assistido por computadores está resumida na seqüência de figuras (a) a (c) na fig. 3.

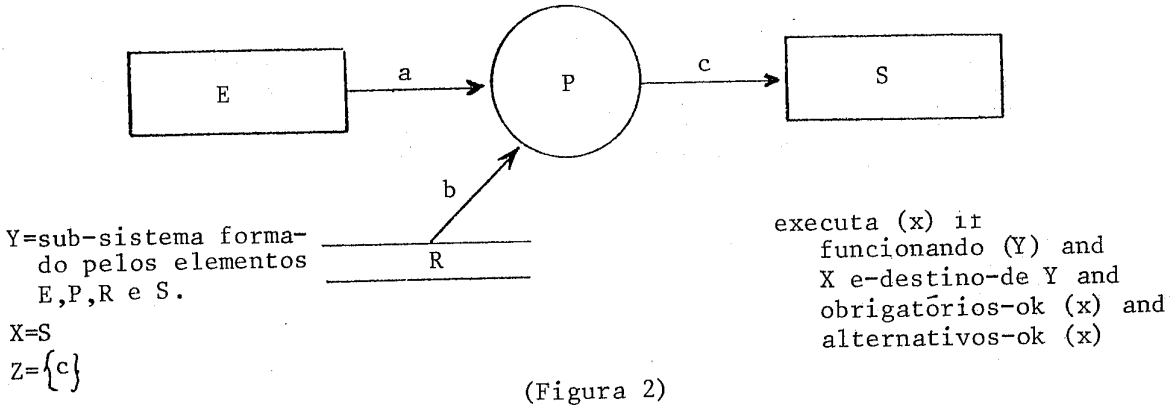
Na primeira (como no sistema UNIX), ferramentas

"naturais" de programação agrupam-se em torno de um sistema operacional, que por estar descrito em linguagem de alto nível, permite generalidade de dados (representações internas de arquivos são abstraídas) e de programas (todas as ferramentas supõem a manipulação de programas com a mesma representação). A grande ferramenta conceitual dos ambientes do tipo (a) são os analisadores léxicos e sintáticos e seus processadores (ex: compiladores de compiladores). Se, nem mesmo qual linguagem de programação deva ser usada, é um problema central no caso (a), este é o grande elemento integrador dos ambientes tipo (b). No caso (b), linguagens de programação de nível alto e muito alto, são os elementos centrais do ambiente de desenvolvimento. Servem de exemplos da situação (b) os ambientes que estão sendo construídos em torno de ADA e Modula-2 (ex: APSE) e os ambientes já produzidos em torno de LISP (ex: INTERLISP e COMMON LISP). A terceira situação, caso (c) requer que as ferramentas de desenvolvimento gravitem em torno de métodos de desenvolvimento. Segundo a orientação dos trabalhos que conduzimos, poderemos definir "esquemas" ou "esqueletos de métodos" (em uma base de conhecimento que disponha de módulos de descrições formais) que seriam integrados para produzir métodos para classes de problemas particulares, que fossem o foco de aplicação para um determinado grupo de engenheiros de software. Estimamos poder definir mapeamentos entre conjuntos de ferramentas padrão de ambientes de desenvolvimento e "módulos" de métodos de nossa base de conhecimentos. Um conjunto típico de ferramentas seria constituído por módulos de funções gráficas que permitiriam o projeto de interface do ambiente para seus usuários. Por exemplo, se o nosso problema devesse ser tratado através da definição de objetos e das mensagens que o ativam, o meta-ambiente imaginado deveria ser capaz de gerar uma versão aproximada do Smalltalk.

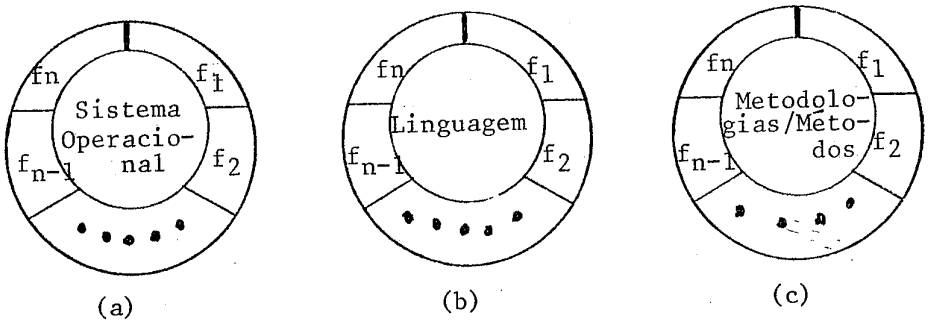
A fase atual da pesquisa descrita procura analisar métodos e metodologias de análise de requisitos em busca de padrões para a solução de problemas na área de desenvolvimento de software.



(Figura 1)



(Figura 2)



(Figura 3)

Referências

(1) Bohem,B.,"Software Engineering", IEEE Trans.Comp.,vol 25, nº 12, dez.1976.

(2) Lucena,C.J.,"Curso Avançado em Engenharia de Software", versão preliminar, março 1985.

(3) Balzer,R. et al, "Software Technology in the 1990's: Using a New Paradigm", in Proceeding of Software Process Workshop,Surrey, fev.1984.

(4) De Remer,F. e Kron,H.,"Programming-in-the-large Versus Programming-in-the-Small", IEEE Transactions Software Engineering SE-2, junho 1976.

(5) Rich,E. "Artificial Intelligence", McGraw Hill, 1983.

(6) Kowalski,R."Software Engineering and Artificial Intelligence",SPL Insight Award Lecture, maio 1984.

(7) Simplício Filho,F.C. "Análise Executável - Um Médoto de Análise de Sistemas Baseado em Representação de Conhecimento", Proposta de Dissertação de Mestrado, PUC/RJ, julho 1985.

(8) Ince,D.C."Module Interconnection Languages and PROLOG", Sigplan Notices, agosto 1984.

(9) Hammond,P. e Sergot,M. "APES: Augmented PROLOG for Expert Systems", Reference Manual, micro-PROLOG version, 1st ed., Logic Based Systems Ltd, England, july 1984.

(10) Lins de Carvalho,R. et al "SAFO: Sistema Automático de Formalização do Conhecimento", Manual de Utilização, PUC/RJ,1984.

(11) Soares, J.F., "Uma Ferramenta de Programação em Lógica para a Construção de Protótipos de Sistemas a partir de suas Especificações Funcionais",Tese de Mestrado,PUC/RJ, agosto 1985.

(12) Jackson,M.A."Principles of Program Design", Academic Press, 1975.

(13) Jackson,M.A."System Development", Prentice-Hall, 1983.

(14) Lehman,M.M. et al "Another Look at Software Design Methodology", ICST Doc.Res.Rep.83/13, abril 1984.