* RELATÓRIO TÉCNICO *

WALK: UM PRODUTO DA INTELIGÊNCIA ARTIFICIAL APLICADA À ENGENHARIA DE SOFTWARE

Marcos Gonzalez de Souza
Claudia Lage Rebello da Motta

NCE 37/90

Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
Caixa Postal 2324
20001 - Rio de Janeiro - RJ
BRASIL

* Este artigo foi publicado originalmente nos Anais do XXIII Congresso Nacional de Informática da SUCESU, Rio de Janeiro, em Agosto de 1990.

TÍTULO: WALK: Um Produto da Inteligência Artificial Aplicada à Engenharia de Software

AUTOR: Marcos Gonzalez de Souza

ENDEREÇO: Núcleo de Computação Eletrônica – UFRJ
Caixa Postal 2324 – CEP 20001 Rio de Janeiro – RJ – Brasil
E–mail: ncd01003@ufrj.bitnet
Tel. (021) 290.3212 (ramal 327)

AUTOR: Claudia Lage Rebello da Motta

ENDEREÇO: Núcleo de Computação Eletrônica – UFRJ
Caixa Postal 2324 – CEP 20001 Rio de Janeiro – RJ – Brasil
E–mail: ncd01002@ufrj.bitnet
Tel. (021) 290.3212 (ramal 327)

CURRICULUM VITÆ: Marcos Gonzalez de Souza é Bacharel em Matemática Aplicada, modalidade Informática, pela UFRJ; mestrando em Ciência da Computação pela COPPE/UFRJ; pesquisador do Grupo de Inteligência Artificial da UFRJ (GIA/UFRJ)

TOTAL DE PÁGINAS: 18

RESUMO:

Este trabalho pretende fazer uma revisão da utilização da Inteligência Artificial na Engenharia de Software em seus diversos níveis de aplicação, focalizando com maior cuidado o sistema WALK, em desenvolvimento pelo GIA, que está inserido no tema abordado no artigo.

PALAVRAS–CHAVES:

Inteligência Artificial
Sistemas Especialistas
Engenharia de Software
Projeto Estruturado de Sistemas

Este Projeto foi parcialmente financiado com recursos do Projeto ESTRA da SID-Infomática

WALK: Um Produto da Inteligência Artificial Aplicada à Engenharia de Software

O trabalho pretende fazer uma revisão da utilização da Inteligência Artificial na Engenharia de Software em seus diversos níveis de aplicação, focalizando com maior cuidado o sistema WALK, um sistema especialista em Projeto Estruturado, em desenvolvimento pelo GIA/UFRJ, que está inserido no tema abordado.

WALK: A Result of Artificial Intelligence Technology applied to Software Engineering

This report is a survey of the use of Artificial Intelligence technology on Software Engineering and its different levels of applications. It also focus on the WALK, an Expert System on Strutured Design that is an example of this subject, which is under development by GIA/UFRJ (AI Research Group).

1. Introdução

A década de 70 ficou caracterizada, dentro da informática, pela revolução tecnológica provocada, principalmente, pelo desenvolvimento da microeletrônica, a partir do advento de novas tecnologias de fabricação e descoberta de novos materiais, que permitiram fundir cada vez mais dispositivos eletrônicos de alta capacidade a um custo cada vez menor, resultando, então, num crescimento exponencial do desempenho do hardware.

O desenvolvimento do Software, no entanto, não teve fôlego suficiente para acompanhar a evolução do Hardware, fato talvez decorrente de sua própria natureza lógica. A Engenharia de Software, também surgida no seio da década de 70, carecia de uma formalização mais precisa dos processos de software. Foram criados, então, os modelos e os métodos, na busca de uma melhora da qualidade do produto, do crescimento da produtividade e, finalmente, na sistemática produção e manutenção de software.

No início dos anos 80, apareceram as estações de trabalho de 16 e 32 bits, e com elas, os primeiros ambientes de desenvolvimento de software. Estes ambientes, que resumiam-se numa aplicação das metodologias criadas na década anterior, trouxeram um novo incentivo às pesquisas na área, já que automatizavam grande parte da metodologia, em geral a parte mais braçal, mas esbarravam mais uma vez no cerne da questão da Engenharia do Software: [FAIR85] "*Software não tem massa, volume, cor nem odor, ou seja, não tem propriedades físicas!*". Em outras palavras, a Engenharia de Software é uma ciência não exata, ao contrário das outras *engenharias*, que constróem pontes, navios, carros, etc., simplesmente porque sua prática está baseada em conceitos abstratos, não estando, portanto, sujeita a leis rígidas.

Para contornar tal situação, a comunidade científica ligada à Engenharia de Software vem buscando, atualmente, soluções em, principalmente, duas frentes: a formalização de métodos e a incorporação de técnicas da Inteligência Artificial (IA) aos processos de desenvolvimento de software. Neste artigo, procuraremos dar uma visão geral do uso de algumas técnicas de IA que vem sendo usadas na Engenharia de Software, ressaltando o projeto WALK, um sistema especialista em Modularização de Projetos Estruturados, atualmente em desenvolvimento no NCE/UFRJ.

2. Engenharia de Software

Engenharia de Software é uma disciplina relativamente nova. Nesta seção serão revistos alguns dos principais aspectos desta ciência, seus problemas e suas perspectivas, sem nos determos, no entanto, muito profundamente.

Existem várias definições do que seja a ciência da Engenharia de Software. Usaremos aqui uma delas:

"*Engenharia de Software é a disciplina tecnológica e gerencial voltada para a sistemática produção e manutenção de produtos de software, desenvolvidos e modificados em tempo e custos estimados*" [FAIR85]. A produção e manutenção citadas na definição estão intimamente ligadas com a noção de um modelo do processo de software conhecido como seu Ciclo de Vida.

O Ciclo de Vida Convencional do Software (Fig. 1)

Este modelo conceitual, formalizado no início da década de 70, é uma representação do processo de software e pode ser visto, numa primeira aproximação, em duas fases principais: a fase de desenvolvimento e a fase de operação e manutenção. O desenvolvimento inicia quando a necessidade por um produto é identificada, e termina quando sua implementação estiver testada e liberada para operação. A operação e manutenção incluem todas as atividades durante a operação do software, tal como correções, adaptações do sistema a seu ambiente, etc.

A fase de desenvolvimento, por sua vez, é dividida nas seguintes sub–fases: Requisitos, Especificações, Projeto e Implementação (Não confundir aqui a fase de Projeto com as metodologias de Projeto de Sistemas, que serão vistas mais adiante).

Os requisitos do Software devem ser dados pelo usuário final ou, no caso do software estar incluído num sistema maior, pelos requisitos impostos por este sistema. Os requisitos incluem todos os contextos em que o software possa vir a ser inserido, a expectativa do seu funcionamento e seu escopo. Com estas informações, os projetistas do software tentam entender os requisitos e esboçam as especificações do software, que descrevem O QUE deve ser feito, sem se preocuparem COMO, cuidando principalmente da adequabilidade, omissões, inconsistências e ambigüidades. Os requisitos de um software de dimensões razoáveis podem ser de difícil especificação, por causa da variedade de funções e da necessidade de expressar suas propriedades, que podem estar implícitas. Em geral, os usuários não sabem o que querem; quando sabem, eles não precisam daquilo que querem [KOWA84].

Especificados os requisitos, dá–se início à fase de projeto. Esta fase descreve COMO o sistema deve ser implementado. Para tanto, os sistemas complexos são decompostos funcionalmente em módulos e suas interações, até que cada módulo possa ser encarado como um micro–universo do todo, só que mais manuseável e de mais fácil implementação.

Durante a fase de Implementação, cada módulo é codificado numa linguagem adequada. Uma liguagem de nível mais alto deve ser usada para a gerência do software. Em seguida, inicia–se o estágio de teste e validação dos programas e suas integrações, entre si e com o exterior.

A última fase, a de manutenção, é considerada, atualmente, a mais crítica da vida de um software. É aqui que os erros cometidos nas fases ateriores são efetivamente corrigidos, só que a um custo muito mais elevado. Quanto mais tarde for detectado um erro, mais aumentam os custos para repará–lo, senão vejamos: os erros advindos das fases de Requisitos e Projeto do Software são cerca de 100 vezes mais caros que os da fase de Implementação! [RAMA84]

Dentro deste quadro, diversas metodologias foram criadas para apoiar as diversas fases do processo, sendo as mais conhecidas as metodologias estruturadas propostas por diversos autores [JACK75, GANE79, YOUR79].

O modelo convencional de Ciclo de Vida de um Software, no entanto, continuava apresentando alguns problemas clássicos, se conduzido manualmente: Requeria um tempo excessivamente grande para desenvolvimento, havia pouco intercâmbio com o usuário final, o que acarretava em gastos futuros e falhava quanto ao reaproveitamento de sistemas já existentes.

Começaram a surgir algumas alternativas que viessem a amenizar o uso do modelo para desenvolvimento de software, sendo talvez as mais importantes a técnica da Prototipação Rápida e o uso de linguagens de nível muito alto para descrição de funções. [RAMA84]

A primeira visa criar, de forma rápida, uma *amostra* do software a ser produzido, ou seja, um protótipo, que seja facilmente manuseado, alterado e transportado. Este protótipo servirá como interface dinâmica entre o engenheiro e o usuário. Em geral, ele é desenvolvido *de baixo para cima*, isto é, pela funções mais simples para as mais complexas, não contém todas as especificações do sistema e os módulos mais usados podem ser encontrados numa biblioteca de rotinas. Alguns autores propõem uma Prototipação Evolutiva, que substituiria por completo o modelo de desenvolvimento convencional; já outros autores dizem que a prototipação não visa substituir nenhuma metodologia, e sim dinamizá–la. De qualquer forma, trata–se de uma técnica alternativa.

O uso de linguagens de nível muito alto, ao mesmo tempo expressivas e simples, para escrever programas funcionais permite sua codificação automática e a geração de programas de aplicação. Desta forma, programadores podem geram sistemas com poucos erros. No entanto, torna–se claro que a geração de programas funcionais dependem da aplicação. Esta deve atender às expectativas de que suas funções são bem conhecidas (como por exemplo: aplicações bancárias, folha de pagamento, etc.)

2.1. O Desenvolvimento de Software Automatizado

A necessidade de aumentar a produtividade no desenvolvimento de software e a qualidade do produto fez a Engenharia de Software recorrer à automação dos processos de desenvolvimento. Embora não esteja resolvido o problema de definição de modelos de processo adequados para a aplicação de métodos automatizados, muita coisa já foi feita tomando–se ainda como referência o modelo em fases.

Um ambiente de Desenvolvimento de Software é definido como "*Uma coleção coordenada de ferramentas de software organizada para dar apoio a algum enfoque para o desenvolvimento de software em conformidade com algum modelo do processo de software*" [LUCE87], e podem ser classificados da seguinte maneira:

a) Ambientes para programação;

b) Ferramental (*toolboxes*) – coleção de ferramentas que fornecem apoio ao trabalho em equipe;

c) Ambientes Integrados ou CASEs (Computer–Aided Software Engineering) – orientados para integração de diversas fases do modelo de ciclo de vida;

d) Repositórios – permitem o armazenamento e recuperação de descrição de software.

Atualmente, pode–se encontrar algumas ambientes com aquelas características no mercado. No entanto, o que se observa é que a simples automação não vem produzindo os resultados esperados. Os casos que envolvem aspectos mais subjetivos do conhecimento, como por exemplo decisões sobre o projeto, requerem uma maior participação do ambiente, de forma a aliviar a carga do analista desta atividade, que ainda é altamente dependente de sua experiência no uso da própria metodologia. Além disso, as metodologias usadas na Engenharia de Software – mais uma vez comparando com outras engenharias, onde as medologias levam a uma boa solução – não garantem a produção de softwares absolutamente confiáveis, já que permanecem susceptíveis a erro.

3. Onde Entra a Inteligência Artificial?

O projeto de Computadores de Quinta Geração Japonês indentificou a Importância da aplicação da Inteligência Artificial, principalmente quanto ao uso das linguagens baseadas em lógica, trazendo assim uma esperança técnica para a crise de software predita e uma revolução dos conceitos da ciência da computação, o que influiu diretamente no desenvolvimento da IA como nova tecnologia de software e, em seguida, de desenvolvimento de software [KOWA84]. Daí para as primeiras aplicações de IA na Engenharia de Software não demorou muito.

Os trabalhos de IA aplicada à Engenharia de Software expandiram–se em duas frentes principais: Uma, no sentido da psicologia cognitiva, onde vem sendo investigado um

meta–modelo de como especialistas realizam algumas tarefas para implementar um modelo num programa; uma outra, aplicando as técnicas da IA nos métodos já conhecidos da Engenharia de Software, também conhecidos como *softwares assistentes baseados no conhecimento*.

Nesta seção, veremos as aplicações que estão em desenvolvimento atualmente nestes campos. Didaticamente, a seção foi dividida em sub–seções que agrupam *temas* da Engenharia de Software onde IA vem sendo aplicada.

3.1. IA Aplicada às Fases do Ciclo de Vida e às Metodologias

O projeto de um software é frequentemente delegado a um engenheiro de software especialista. Ele é responsável por interpretar os requisitos e especificar uma solução para o novo software, e para fazê–lo, são necessários conhecimento e experiência no desenvolvimento de projetos similares. Técnicas de IA podem ser incorporadas em ambientes de desenvolvimento para facilitar a especificação e projeto.

Um outro aspecto a ser considerado é o fato de o Ciclo convencional de Vida de um software acarretar num processo bastante lento no desenvolvimento do produto. IA pode ser usada para dinamizar o processo, se aliada a outras técnicas provindas da própria Engenharia de Software.

Além disso, pode–se usar IA para promover a interrelação das diversas fases do Ciclo de Vida, gerenciando suas dependências e *feedbacks*.

Veremos a seguir, com mais detalhes o uso de técnicas de IA nas fases do Ciclo de Vida, incluindo as técnicas alternativas.

3.1.1. Fases de Requisitos e Especificação

O principal objetivo de uma linguagem que dê suporte ao levantamento de requisitos e especificações de um software é promover facilidades para reunião e representação de conhecimento de um domínio real de um modo conveniente e natural e, ao mesmo tempo, que organize e estruture este conhecimento de forma a torná–lo de fácil entendimento, tanto pelo projetista, quanto pelo usuário final.

Para tanto, esta linguagem deve permitir a descrição de entidades e eventos reais, dentro do domínio, assim como suas excessões, que, em outras palavras, pode ser descrita como uma linguagem orientada para o domínio. Em geral, as metodologias usadas falham no tratamento da última propriedade. E são as excessões que caracterizam aspectos de um domínio real.

Um outro aspecto desta linguagem é que ela deve suportar o processo de abstração, comumente usado nas metodologias tradicionais, que separa os pontos mais importantes e introduz os demais com sucessivos passos de refinamento.

As atuais pesquisas buscam uma linguagens com as características descritas. Em geral, os caminhos levam à uma combinação de metodologias tradicionais com uma linguagem voltada para lógica. Esta linguagem permitiria que a especificação pudesse ser analisada e *executada*, antes que viesse a ser transformada numa especificação funcional, um projeto de software ou um programa, o que diminuiria os mal-entendidos entre o analista e o usuário. (ver [KOWA84],[RAMA84],[BORG85],[LUBA86])

3.1.2. Fase de Projeto

As atividades da IA associadas ao Projeto de software estão basicamente voltadas para dois pontos: o raciocínio por trás das decisões do Projeto e o desenvolvimento de sistemas inteligentes, baseados em metodologias já desenvolvidas, que assistem ao projetista no desenvolvimento do software.

O primeiro ponto está diretamente relacionado com a busca por novas metodologias, ferramentas e, mesmo, novos modelos da atividade de projetar. A IA está particularmente interessada nos caminhos traçados pelo raciocínio de um projetista especialista em comparação com um projetista junior, tentando responder as questões relevantes: Como pensa o projetista? O que acontece quando ele se depara com um domínio não conhecido? Quais as heurísticas usadas para a seleção de um método? Bons trabalhos neste ramo de pesquisa são descritos em [ADEL85],[KANT85],[STEI85]

Os assistentes especialistas em metodologias são ferramentas (que incluem, em geral, Sistemas Especialistas associados) que ensinam um projetista a usar as regras de uma metologia. São duas as formas de realizar esta tarefa: uma, onde o projetista trabalha diretamente sob a supervisão do assistente, ou então ele submete seu projeto para apreciação do assistente. De ambas as formas, o projetista pode obter orientação sobre a aplicação do método, alternativas para solucão de problemas ou até mesmo planejamento e controle da solução. [LUCE87] e [GANE88] apresentam ferramentas com as características apresentadas. Na seção 4, discursaremos um pouco sobre o WALK, que também se enquadra nesta categoria.

3.1.3. Técnicas Alternativas

IA também pode ser encontrada nas técnicas alternativas para o desenvolvimento de software, já descritas anteriormente ([GLAS88],[LUBA86],[KOWA84]).

IA pode assistir o desenvolvimento de rápidos protótipos de duas formas: usando técnicas de programação automática para gerar protótipos de implementações sem se concentrar na eficiência, tolerância à falhas, generalidade ou outros detalhes de implementação, o que pode ser considerado um problema simples de geração automática. Ou então, é possível gerar especificação numa linguagem própria, executável, resultando numa especificação operacional. Prolog pode ser esta linguagem. Além disso, um grupo de pequenos Sistemas Especialistas podem ser usados na Prototipação, pois atende a todas as características desta técnica, com a vantagem de que desta forma, é possível apresentar exemplos mais próximos da realidade do usuário.

Uma outra técnica alternativa é a da reutilização de código ou partes do projeto de um sistema. O princípio por trás do uso da IA nesta técnica é que há conhecimento e experiência imbutidos nos softwares outrora produzidos. É possível produzir Sistemas Especialistas para codificar esquemas de projeto e planos de programa (métodos padrão para tratar uma tarefa), incorporando a eles bibliotecas de referências para acesso aos códigos destes esquemas e planos.

3.2. IA Aplicada à Programação

Na área de programação encontram–se dois grandes campos de pesquisa da IA: a programacão automática e as ferramentas inteligentes de apoio à programação.

A programação automática é uma das áreas mais antigas de interesse da IA, sem que se tenha conseguido atingir resultados satisfatórios. Entende–se aqui por programação automática a tarefa em que, ao se obter uma especificação de alto nível, traduzí–la (leia–se compilá–la), obtendo um programa fonte ou uma especificação de nível mais baixo que, por sua vez, pode ser automaticamente compilada [BALZ85].

Realizar tal tarefa esbarra em dois paradigmas existentes nos processos de desenvolvimento de software atuais: Primeiro, não existe tecnologia para o gerenciamento de atividades intensamente baseadas no conhecimento, como é a de programação e a dos processos de desenvolvimento. A programação é uma atividade informal, centrada na pessoa (programador). O processo de desenvolvimento de software, que convertem uma especificação em uma implementação eficiente, além de também serem informais e centrados na pessoa (analista), são mal documentadas. Segundo, a manutenção é realizada, geralmente, sobre os fontes dos programas que são, pela própria natureza do bom programador, otimizados, e essas otimizações escondem informações. Esses paradigmas são, concomitantemente, estímulos para o desenvolvimento da programação automática.

As pesquisas que vem sendo realizadas neste campo buscam realizar a tarefa de compilação das especificações através refinamentos baseados em regras [REYN86] ou técnicas de

transformação [FICK85] para automatizar tal processo. O modelo geral de transformação parte de uma especificação formal que é submetida a sucessivas transformações, gerando, a cada passo, novas especificações. A inteligência do processo está, no caso, inserida nestas transformações.

O outro campo de atuação da IA aplicada à programação está nas ferramentas inteligentes de apoio à programação. Estas ferramentas vão desde utilitários inteligentes (compiladores especialistas, que gera códigos intermediários ou que utilizam pouca memória, os depuradores especialistas, para a localização e depuração de erros que normalmente seriam detectados em tempo de execução, tradutores de programas fontes, que transformam fontes produzidos em uma linguagem para outra, etc) até os softwares que analisam e entendem programas ([JOHN85],[WATE81]) ou que testam programas [ZUAL86].

3.3. IA Aplicada à Interface Homem–Máquina

O desenvolvimento de software cada vez mais complexos traz, hoje em dia, um problema de sua sub–utilização, pois seu entendimento completo depende de uma absorção das informações contidas em manuais, tão quão complexos e volumosos. A utilização dos tradicionais *helps* provou não ser a ideal.

As características fundamentais que uma interface inteligente deve conter são: ser *amigável*, ter um esquema de janelas sucessivas, um módulo de explicações e apontar erros ou mesmo mal uso da ferramenta, de forma interativa. A idéia é de que o sistema tenha o conhecimento de todas as capacidades da ferramenta, bem como de quais comandos são definidos a partir de outros, podendo, assim, otimizar o trabalho do usuário, além de ensinar–lhe outras formas de obter o mesmo efeito final.

Nestas aplicações, podemos encontrar alguns campos de pesquisa de IA, sendo Hipertextos e Processamento de Linguagem Natural os principais.

3.4. IA Aplicada à Gerência de Banco de Dados

Se compararmos os atuais sistemas de gerenciamento de Banco de Dados às necessidades das modernas aplicações, tais como suporte à decisões de negócios, deparamos com alguns problemas. Veremos como IA, particularmente Sistemas Especialistas, tem sido usada para minimizar tais problemas.

Os problemas citados estão principalmente ligados às linguagens de consulta a Banco de Dados. Em primeiro lugar, há o fato do crescimento do usuários em potencial para aplicações de grandes Banco de Dados, que requerem linguagens de mais alto nível para que possam obter informações gerenciais sem precisarem conhecer mais a fundo a ciência da computação. Estes

usuários também começam a carecer de operações mais complexas que a serem realizadas sobre dados que não são providos pela linguagem de consulta, forçando–os a usarem linguagens de programação para gerenciamento de Banco de Dados. E as linguagens deveriam ser capazes de perceber quando uma consulta feita pelo usuário depende implicitamente de uma outra consulta, e fazê–las. E as linguagens deveriam ser capazes de atender consultas/atualizações simultâneas de múltiplos usuários.

Os Sistemas Especialistas têm sido usados para a produção de Bancos de Dados inteligentes, tanto quanto à recuperação de informações, introduzindo capacidade dedutiva, como quanto à sua implementação (otimização, segurança, integridade, etc.) (ver [JARK84])

3.5. IA Aplicada à Geração Automática de Ambientes de Desenvolvimento de Software

A geração de meta–ambientes para a producão de software é justificada pelo seguinte fato [LUCE87]: diferentes áreas de aplicação e até mesmo diferentes projetos particulares de software requerem diferentes ambientes para sua concepção e desenvolvimento, o que não é economicamente viável. Os meta–ambientes encontram–se em fase artezanal, de forma que não há ainda muitas definições de como devem ser produzidos e que modelos devem ser usados. No entanto, já é possível classificá–los:

Meta–ambientes com estruturas puramente gerativa: Possuem como entradas a sintaxe abstrata da(s) linguagem(ns), a sintaxe concreta e os mecanismos de projeto de ferramentas, tendo como saída o ambiente centrado na linguagem.

Configuradores de ambientes baseados no modelo do processo de software e das aplicações: Possuem como entradas os sistemas de configuração de núcleo e ferramentas e a definição de processo de software, inserem na sua arquitetura um núcleo somado aos módulos de ferramentas genéricas e configuráveis, tendo como saída um ambiente orientado para classes de aplicações.

Ambientes exploratórios para a produção de ambientes centrados em métodos e metodologias: Possuem como entradas metodologias, heurísticas e conhecimento sobre a área de aplicação, inserem na sua arquitetura um núcleo para o projeto interativo de ambientes e um módulo de ferramentas genéricas e configuráveis, tendo como saída um ambiente baseado em metodologias.

De uma maneira geral, os meta–ambientes chamados de puramente gerativos permitem a criação de ambientes em torno de uma linguagem de programação escolhida, que é estendida para permitir programação no nível de módulos. Os suportes de ambientes que permitem a

configuração de um ambiente particular para um ciclo de vida dado, ao qual são associados conjuntos de ferramentas, favorecem, mais livremente, a utilização de diferentes metodologias, técnicas e linguagens. Em ambos os casos, é possível lançar mão de técnicas de IA para aumentar o poder dos ambientes, lançando mão, por exemplo, dos ambientes anteriormente descritos.

4. O Sistema WALK

Passaremos agora ao segundo objetivo deste artigo, que na verdade está inserido dentro do primeiro, que é descrever o WALK, um sistema especialista escrito em PROLOG que verifica a qualidade de um Projeto de sistema segundo a metodologia estruturada, inicialmente proposta por Larry Constantine e posteriormente endossada por outros autores, a partir da descrição do Diagrama Hierárquico de Módulos (DHM).

4.1. Projeto Estruturado

O Projeto Estruturado tem como objetivo construir programas como estruturas de módulos independentes e unifuniconais. A sua eficácia está relacionada com uma característica de resolução de problemas em geral: a solução fica mais difícil quando todos os aspectos do problema são considerados simultaneamente. É mais fácil se o problema puder ser resolvido por partes.

Paralelamente aos benefícios que esta metodologia apresenta, ela traz consigo dois problemas (já vistos): ela é trabalhosa e requer uma certa experiência do analista, que necessita de um conhecimento razoável sobre a metodologia.

O DHM é a representação gráfica dos módulos e interfaces entre eles no Projeto Estruturado. Nem o DHM nem a especificação dos módulos por si só mostram a qualidade de um Projeto. É preciso levar em conta outros aspectos da metodologia. Os principais fundamentos do Projeto Estruturado é o fato de grandes sistemas poderem ser particionados em blocos mais manuseáveis, chamados módulos. Este particionamento, contudo, deve ser feito segundo alguns critérios, sendo os mais importantes o critério da Coesão e o do Acoplamento. A Coesão mede a funcionalidade de cada módulo, e o Acoplamento, o grau de dependência entre os módulos. Os princípios básicos são que quanto maior a Coesão de um módulo, mais funcional ele é, e quanto mais funcional for um módulo, melhor; e quanto menor for o Acoplamento entre dois módulos, menos dependente um módulo é do outro.

O WALK tem como objetivo auxiliar o projetista na tarefa de melhorar a qualidade do Projeto.

4.2. O WALK como Assistente Especialista na Metodologia de Projeto Estruturado

O WALK foi projetado para funcionar sob um ambiente para desenvolvimento de sistemas. No caso, o ambiente escolhido, o PC–CASE, da Base Tecnologia [BASE89], é composto de diversas ferramentas integradas, e fornece suporte para todas as fases da análise e projeto de sistemas, o que inclui a criação de diagramas, a verificação da correção do projeto e a geração automática da documentação na forma de diagramas e relatórios.

O WALK foi escrito em PROLOG e roda em IBM–PC ou compatíveis. Constatou–se que o uso dessa linguagem foi bastante adequado para a implementação do sistema, já que o DHM pode ser visto como um problema de grafos, contendo hierarquias e relações.

A utilização do ambiente segue os seguintes procedimentos: o projetista desenvolve o seu projeto utilizando a ferramenta e submete–o à apreciação do WALK. Este, por sua vez, oferece 3 (três) tipos de avaliação (níveis de avaliação), que diferem entre si pelo grau de complexidade dos problemas que abordam. O primeiro nível faz um levantamento dos erros chamados *primários* que, em geral, são erros que podem ser cometidos por iniciantes ou por projetistas que possuem pouco conhecimento sobre a metodologia. O que este nível de avaliação faz, na verdade, é *limpar* o projeto.

O segundo nível de avaliação trata dos problemas cruciais da metodologia. É nesta hora que a Coesão e o Acoplamento dos módulos, entre outras diretrizes, é abordada. Finalmente, o terceiro nível visa buscar melhorias e otimizações dentro do Projeto.

São características desta abordagem que um nível só pode, obrigatoriamente, ser percorrido se todos os anteriores já tiverem sido dados como satisfeitos pelo sistema, mas o usuário percorre os níveis se ele assim o desejar, ou seja, o projetista fica à vontade de não submeter seu projeto ao sistema, ou submeter apenas ao primeiro nível, aos dois primeiros ou a todos. Ao final de cada avaliação, o sistema aponta erros cometidos e sugere alternativas de mudanças no projeto, que o usuário, da mesma forma, pode aceitá–las ou não.

Este processo acima descrito deve ser repetido a cada mudança no Projeto, até que ou o usuário ou o sistema se dê por satisfeito.

A integração da ferramenta CASE com o sistema especialista (Fig. 2) se dá através de 3 (três) entradas:

a) A descrição do Projeto e Dicionário de Dados, através de um arquivo gerado pela ferramenta, em linguagens criadas para tais fins;

b) Perguntas ao usuários, a respeito de aspectos peculiares a seu Projeto, e que o Sistema Especialista não tem acesso mecanicamente;

c) Descrição em linguagem de alto nível do código de cada módulo, que dá subsídio ao WALK sobre o que cada módulo faz e deveria fazer.

Um detalhe importante é que o sistema foi projetado de forma que não seja dependente da ferramenta. O ambiente é gerenciado por um programa que está acima da ferramenta e do sistema especialista, fazendo a interface entre ambos.

4.3 Estado atual do Sistema

O estado atual não atende à última especificação citada (letra c). A descrição, por enquanto, é dada através de perguntas ao usuário (letra b). Além disso, fazem parte dos planos da equipe de desenvolvimento prover ao ambiente uma integração maior, de forma que a avaliação do WALK seja dada através da própria ferramenta, tornando–o mais interativo (leia–se *on–line*).

5. Conclusões

Notamos que o uso da IA aplicada à Engenharia de Software, apesar de tudo, ainda é controvertido. Deixaremos isso claro, apresentado quatro opiniões, três provindas de autores consagrados e uma nossa. As três opiniões colhidas mostram os três *lados da moeda*: a descrença, a cautela e o otimismo.

A primeira opinião é expressa por David Parnas em [PARN85]. Neste artigo, que é, na verdade, uma carta endereçada a James Offut, diretor assistente da SDIO (Strategic Defense Initiative Organization), responsável pelo gerenciamento do suporte computacional para guerra, Parnas pede seu desligamento daquela organização e explica os motivos. Trata–se de uma relatório contendo seus pontos de vista sobre tal organização, sobre a Engenharia de Software e a Inteligência Artificial. Para resumir, ressaltaremos apenas algumas de suas frases.

"*Devido à enorme demanda de sistemas e nossa impossibilidade de testá–los, nós nunca estaremos aptos a acreditar que nós tivemos sucesso*" (falando a respeito de softwares produzidos para uma guerra nuclear)

"*Os métodos usados atualmente não são adequados para o desenvolvimento de grandes softwares de tempo–real. Uma drástica inovação dos métodos é necessária*"

"*Eu não espero que antes de 20 anos de pesquisa mudaremos este quadro*" (falando dos recursos atuais da Engenharia de Software)

"*Inteligência Artificial tem a mesma relação com inteligência que uma flor de plástico tem com as flores. (...) IA não oferece nenhuma tecnologia mágica para resolver nossos problemas. Técnicas*

heurísticas não resultam sistemas que possamos confiar" (falando sobre IA e o que ela pode fazer pela Engenharia de Software)

"*Um dos problemas básicos com o SDIO é que nós não temos informações suficientes para escrever especificações que possamos confiar. Nesta situação, Programação Automática não oferece nenhuma ajuda*" (falando sobre o uso da Programação Automática no desenvolvimento de software)

A segunda opinião é dada em [ZUAL86]:

"*Para uma aplicação realmente útil de Sistemas Especialistas à Engenharia de Software, entretanto, é preciso pesquisar o raciocínio por trás da resolução de problemas e, particularmente, das metodologias de desenvolvimento de programas*"

"*Devemos ser cautelosos no uso de Sistemas Especialistas na Engenharia de Software. (...) Se a metodologia é bastante específica, não há problemas em se usar Sistemas Especialistas auxiliando na avaliação dos métodos. Mas se a metodologia é fraca, de nada adiantaria o Sistema Especialista*"

"*A maioria das metodologias atuais para desenvolvimento de sistemas de grande porte não são adequadas para admitirem Sistemas Especialistas. No entanto, isto não significa que estes devem ser totalmente ignorados para aplicações na Engenharia de Software*"

A terceira, e mais otimista das opiniões, vem de Jack Mostow, editor convidado da edição da IEEE Transactions on Software Engineering, vol. 11, número 11, de Novembro de 1985, totalmente dedicada à utilização da IA na Engenharia de Software e vice–versa:

"*Uma grande parte das pesquisas realizadas em IA tem relevância direta com a Engenharia de Software. Ambientes de programação da IA estão tornando possível o rápido desenvolvimento de complexas aplicações baseadas no conhecimento que seriam muito mais caras e de difícil manutenção se desenvolvidas com tecnologia convencional.*"

"*As pesquisas em assistentes inteligentes oferecem promissores avanços para a produtividade, a segurança e a flexibilidade de softwares*"

Nossa opinião particular vai mais ao encontro da segunda opinião apresentada. O uso da Inteligência Artificial tem um campo promissor na Engenharia de Software, mas ainda depende de progressos significativos de ambas as ciências. Não somos tão otimistas quanto Mostow, nem tão pessimistas quanto Parnas; sabemos apenas que devagar chegaremos lá.

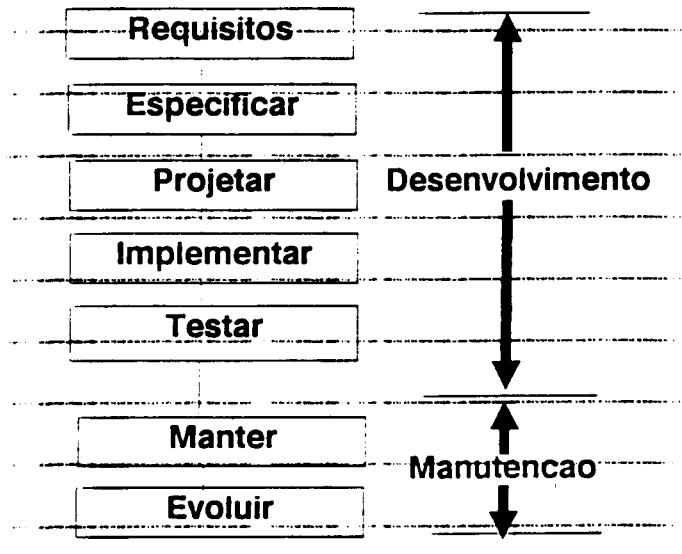
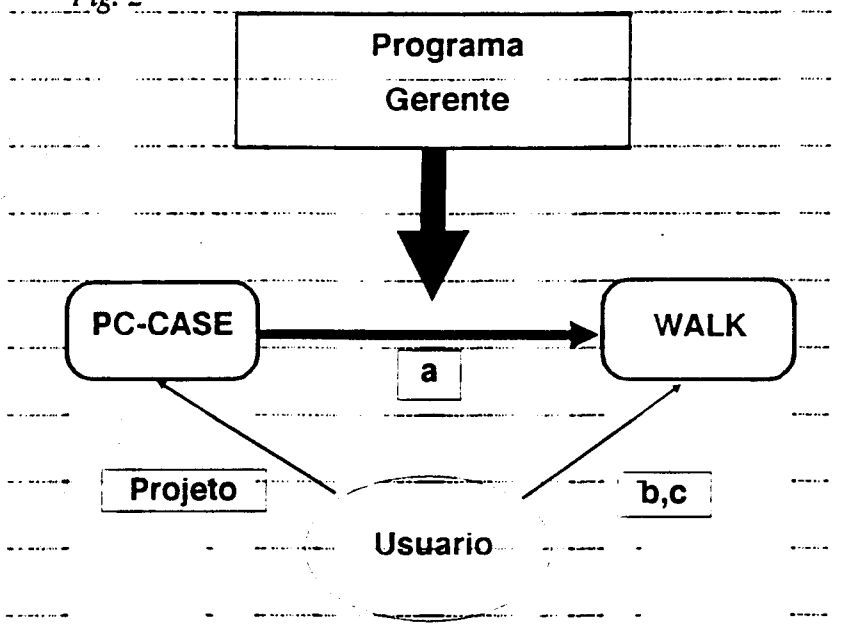


Fig. 1 - Ciclo de Vida do Software

Fig. 2



6. Referências

[**ADEL85**] Adelson, Beth & Soloway, Elliot, *The Role of Domain Experience in Software Design*, IEEE Transactions on Software Engineering, Special Issue on Artificial Intelligence and Software Engineering, Vol. SE-11, Nº 11, pp. 1351-1360, Novembro de 1985

[**BALZ85**] Balzer, Robert, *A 15 Year Perspective on Automatic Programing*, IEEE Transactions on Software Engineering, Special Issue on Artificial Intelligence and Software Engineering, Vol. SE-11, Nº 11, pp. 1257-1268, Novembro de 1985

[**BASE89**] Base Tecnologia Ltda., *PC-CASE Manual do Usuário*, versão 1.0, Fevereiro de 1989

[**BOEH83**] Boehn, Barry W., *Seven Basic Principles of Software Engineering*, The Journal of Systems and Software 3, 1983, pp. 3-24, Elsevier Science Publishing Co., Inc.

[**BORG85**] Borgida, Alexander; Greenspan, Sol & Mylopoulos, John, *Knowledge Representation as the Basis for Requirements Specifications*, IEEE Transactions on Computer, Abril de 1985, pp. 82-90

[**DESM87**] Desmond, John, *Expert Systems in Software Development*, Software News, Julho de 1987, pp. 44-47

[**FAIR85**] Fairley, Richard E., *Software Engineering Concepts*, McGraw-Hill Book Company, 1985

[**FICK85**] Fickas, Stephen F., *Automating the Transformation Development of Software*, IEEE Transactions on Software Engineering, Special Issue on Artificial Intelligence and Software Engineering, Vol. SE-11, Nº 11, pp. 1268-1277, Novembro de 1985

[**GANE79**] Gane, Chris & Sarson, Trish, *Análise Estruturada de Sistemas*, Prentice-Hall Inc., 1979

[**GANE88**] Gane, Chris, *Computer-Aided Software Engineering: The methodologies, The Products, The Future*, Instituto Brasileiro de Pesquisa em Informática - IBPI, Março de 1988

[**GLAS88**] Glasgow, Barry & Graham, Elizabeth, *Rapid Prototyping Using Core Knowledge Bases*, AI Expert, Abril de 1988, pp. 26-35

[**JACK75**] Jackson, M.A., *Principles of Program Design*, Academic Press Inc. Ltd., 1975

[**JARK84**] Jarke, Matthias & Vassiliou, Yannis, *Coupling Expert Systems with Database Management Systems*, Artificial Intelligence Aplications for Business, Proceedings of NYU Sumposiumi, Maio de 1983, pp. 65–85, Ablex Publishing Corporation, New Jersey, EEUU, 1984

[**JOHN85**] Johnson, W. L. & Soloway, Elliot, *Proust: Knowledge-Based Program Understanding*, IEEE Transactions on Software Engineering, vol. SE-11, Nº 3, Março de 1985

[**KANT85**] Kant, Elaine, *Understanding and Automating Algorithm Design*, IEEE Transactions on Software Engineering, Special Issue on Artificial Intelligence and Software Engineering, Vol. SE-11, Nº 11, pp. 1361–1374, Novembro de 1985

[**KOWA84**] Kowalski, Robert, *AI and Software Engineering*, DATAMATION, Novembro 1, 1984, pp. 92–102

[**LUBA86**] Lubars, Mitchell D. & Harandi, Mehdi T., *Intelligent Support for Software Specification and Design*, IEEE Expert, Building Intelligence into Software Tools, Inverno de 1985, pp. 33–41

[**LUCE87**] Lucena, Carlos, *Inteligência Artificial e Engenharia de Software*, Publicações Acadêmico-Cietíficas, PUC-RJ/IBM Brasil, Jorge Zahar Editor Ltda., Brasil, 1987

[**MOTT87**] Motta, Claudia L.R.; Silveira, Pedro M.; Teles, Antônio A.S. & Silva Filho, Ysmar V., *Desenvolvendo Sistema Especialista a Partir de Protótipo*, Anais do XX Congresso Nacional de Informática – SUCESU, vol. II, pp. 882–885, 1987

[**MOTT88**] Motta, Claudia L.R. & Souza, Marcos G., *Uma Proposta de Integração de Software Tipo CASE com Sistemas Especialistas*, Anais do II Simpósio Brasileiro de Engenharia de Software, pp. 113–122, Outubro de 1988

[**PAGE80**] Page-Jones, Meilir, *Projeto Estruturado de Sistemas*, McGraw-Hill, 1980

[**PARN85**] Parnas, David L., *Parnas Resigns from S.D.I.*, University of Victoriai – Canadá, Junho de 1985

[**RAMA84**] Ramamoorthy, C.V.; Prakash, Atul; Wei-Tek Tsai & Yutaka Usuda, *Software Engineering: Problems and Perspectives*, IEEE Transactions on Computer, Outubro de 1984, pp. 191–230

[**REYN86**] Reynolds, Robert G., *PMS: An Inference System to Monitor the Stepwise Refinement of Ada Pseudocode*, IEEE Expert, Building Intelligence into Software Tools, Inverno de 1985, pp. 43–49

[**ROBI83**] Robinson, J. A., *Logic Programming – Past, Present and Future*, New–Generation Computing 1, 1983, pp. 107–124

[**ROLA88**] Roland, Walter G., *A Practical Knowledge*, AI Expert, Abril de 1988, pp. 60–65

[**SILV86**] Silva Filho, Ysmar V.; Teles, Antônio A.S. & Motta, Claudia L.R., *Proposta de Um Sistema Especialista em Projeto Estruturado*, Relatório Técnico NCE0486, Março de 1986

[**STEI85**] Steier, David M. & Kant, Elaine, *The Roles of Execution and Analysis in Algorithm Design*, IEEE Transactions on Software Engineering, Special Issue on Artificial Intelligence and Software Engineering, Vol. SE–11, Nº 11, pp. 1375–1386, Novembro de 1985

[**WATE82**] Waters, Richard C., *The Programmer's Apprentice: Knowledge Based Program Editing*, IEEE Transactions on Software Engineering, vol. SE–8, Nº 1, Janeiro de 1982

[**YOUR79**] Yourdon, Edward & Constantine, Larry L., *Structured Design*, Prentice–Hall Inc., 1979

[**YOUR86**] Yourdon, Edward, *What Ever Happend to Strutured Analysis?*, DATAMATION, Junho de 1986

[**ZUAL86**] Zualkernan, I.; Tsai, W.T. & Volovik, D., *Expert Systems and Software Engineering: Ready for Marriage?*, IEEE Expert, Building Intelligence into Software Tools, Inverno de 1985, pp. 24–31