Qual a prática do desenvolvimento de software?

Jorge Henrique Cabral Fernandes

O software é um produto do trabalho humano cada vez mais presente na sociedade. Qualquer discussão sobre a prática de software deve se fundamentar na compreensão da real natureza do que é software e no relacionamento que ele provoca entre pessoas. Este artigo descreve como o software é um artefato humano que não se enquadra em definições convencionais encontradas no dicionário, pois, além de ser uma entidade de natureza mecânica, é uma entidade descritiva, complexamente hieraquizada, cognitivo-linguística e histórica, concebida através de esforços coletivos durante um considerável período de tempo. Partindo da descrição deste contexto do software, este artigo apresenta uma análise das principais atividades e problemas contemporâneos com os quais se deparam os que desenvolvem, adquirem e usam software e sistemas de computador. Tal análise permite a compreensão do papel central desse artefato humano em nossa sociedade pós-moderna, cujas diversas demandas, expectativas e premissas reforçam cada vez mais o futuro da produção e consumo de bens, produtos e serviços.

O contexto da prática do software A prática do desenvolvimento de software está no cerne de uma relação humana de troca de planos, posses, desejos e necessidades entre três categorias de agentes coletivos: os que usam, os que adquirem e os que produzem software.

Hierarquias de máquinas e usuários Quem usa um software em geral é chamado de **usuário**. O software não é, de fato, uma máquina, mas sim uma descrição de máquina. Ou seja, software é um artefato virtual, incapaz de realizar trabalho a menos que exista uma máquina que carregue e interprete as instruções e informações contidas no mesmo, o que resulta na construção de outra máquina, de ordem superior, com a qual interage o usuário. Em outras palavras, na análise de qualquer sistema de computação estaremos sempre falando de duas máquinas. Uma máquina, de ordem **n**, é a máquina possuída pelo usuário (MPU) antes da carga e interpretação das instruções e informações contidas no software. A outra máquina, de ordem **n+1**, é a máquina com a qual o usuário interage, e que surge quando a máquina de ordem **n** faz a interpretação do software. Da combinação dinâmica entre a MPU e o software surge a máquina de ordem **n+1**, à qual dar-se-á o nome de máquina construída por meio de software (MCSW). O usuário de uma máquina pode ser humano ou máquina, esta última eventualmente atuando como intermediária na relação entre a MCSW e outro humano (usuário final). A figura 1 descreve essas relações. (Veja figura 1)

Linguagens e hierarquias Qualquer que seja a natureza do usuário, para que a relação usuário-máquina se estabeleça de forma efetiva é preciso que exista uma linguagem de conversação com a máquina, exercitada entre o usuário e a MCSW, na qual está definida uma estrutura sintática e semântica para construção de sentenças que permitam a comunicação entre as

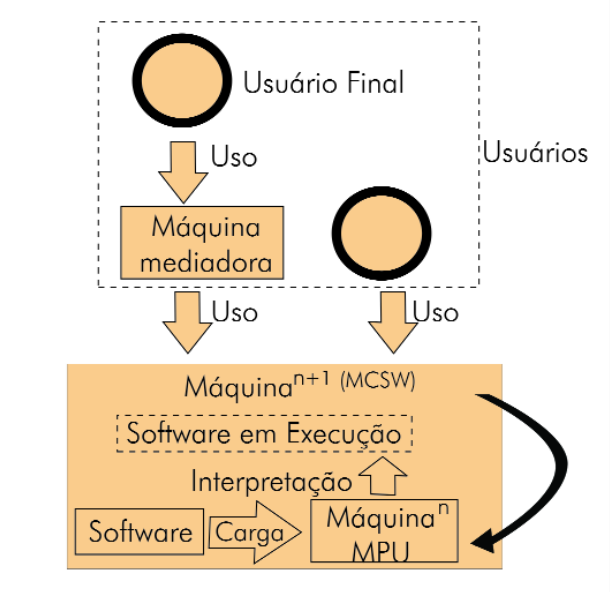

Figura 1 - Categorias de usuários e máquinas

partes: usuário e máquina. Adicionalmente, à MCSW é facultada a capacidade de interagir diretamente com parte ou todo da MPU, o que permite que tarefas da MPU sejam realizadas sob interferência da MCSW. Relacionamentos lingüísticos entre usuários e máquinas podem ser estabelecidos de forma arbitrária e/ou hierarquizada, seja entre o usuário final e a máquina mediadora, entre máquinas independentes, entre a máquina de ordem **n+1** e a máquina de ordem **n**, entre a máquina de ordem **1** e a máquina de ordem **0**, etc.

Pode-se perceber portanto, que a construção de máquinas computáveis é capaz de ser organizada sob diversas formas, principalmente através de uma relação hierarquizada, bastando que, para tal, cada máquina ofereça para a máquina de nível imediatamente superior, um modo de comunicação baseado numa linguagem bem definida, na qual está presente a capacidade de carga e interpretação de planos de construção de máquinas (o software). Eventualmente, na relação hierárquica de mais baixo nível atinge-se a máquina de ordem **0**, construída não mais através de carga e interpretação dinâmica de um software, mas sim através de dispositivos fisicamente imutáveis. Esta máquina **0** é chamada de hardware. A figura 2 apresenta essa relação em maiores detalhes. (Veja figura 2)

O consumidor do software, usualmente chamado de **cliente**, é uma entidade que adquire uma cópia de um software, fornecida por um agente que será chamado de desenvolvedor, através de algum processo de troca, que pode envolver entre outras coisas, dinheiro, bens, ou redes de conhecimento. Do ponto de vista do cliente, o software é visto como um conjunto ordenado de descrições ou instruções, capazes de direcionar a máquina possuída pelo usuário (MPU) para a realização de tarefas que satisfazem às necessidades do último. Como o usuário nem sempre é conhecedor da organização da MPU, o cliente faz a mediação entre o desenvolvedor e o usuário. Sendo assim, o cliente atua antes do uso do software, e é ele que seleciona e decide colocar o software ao alcance da MPU, sobre a qual eventualmente ocorrerá a carga e interpretação do software, criando a MCSW que, espera-se, satisfaça as necessidades do usuário. No

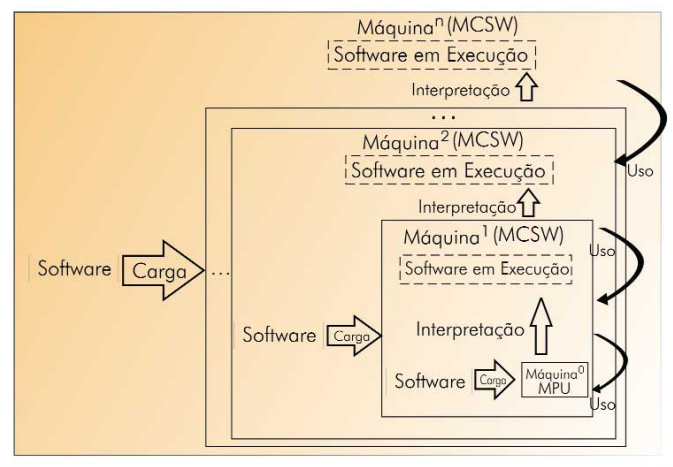

Figura 2 - A natureza hierárquica do software

caso de sistemas de computação de pequena escala, e quando o cliente e usuário são seres humanos, eles tendem a ser a mesma pessoa. Seus papéis se tornam distintos à medida em que qualquer das máquinas, seja a MPU ou a MCSW, se torna complexa e hierarquizada. Em outra situação ocorre que o usuário é uma máquina, sem capacidade de negociação ou decisão.

Dado que o cliente participa de uma relação de troca com o desenvolvedor, com o objetivo de satisfazer as necessidades do usuário, o cliente precisa levar em consideração vários fatores pertinentes, numa relação de produção e consumo de bens.

COMPREENDENDO OS PROBLEMAS, DESEJOS E NECESSIDADES DO USUÁRIO Para poder adquirir o software satisfatório, o cliente precisa compreender: quais os problema e necessidades com as quais o usuário convive e qual a MPU. Definido o contexto da solução, usualmente chamado **domínio da aplicação**, o cliente planeja uma solução para o mesmo através da definição das propriedades de uma máquina necessária para satisfazer ao usuário. Como o cliente não constrói diretamente o software, o plano de solução é expresso através de uma definição de linguagem necessária ao usuário (LNU), com gramática (sintaxe) e lógica (semântica) bem definidas. A LNU, verbalizada pela MCSW, será capaz de se comunicar com o usuário, permitindo-o expressar tarefas a executar, e obter resultados adequados. Outro aspecto que o cliente considera é que o problema e as necessidades do usuário existem no mundo real, e portanto apresentam-se em um contexto de tempo, espaço e recursos limitados. Sendo assim, também faz parte de uma solução satisfatória de aquisição de software: a seleção de um desenvolvedor de software capaz de criar um plano de construção da MCSW, que seja carregável e interpretável por MPU; e a adoção de um conjunto de condições que permitam que o software esteja disponível para o usuário no momento em que este necessitar, e dentro de uma relação de custo-benefício satisfatória para o cliente.

O DESENVOLVEDOR O desenvolvedor de software é um agente coletivo, responsável por criar um plano de construção de máquina (software) que esteja dentro das condições estabelecidas pelo cliente. Software é, portanto, uma meta-máquina. A figura 3 expressa a relação entre os três elementos humanos que participam do cenário de prática do software. (Veja figura 3)

UM EXEMPLO CONCRETO Para tornar mais clara a explanação apresenta-se abaixo um exemplo concreto: o (software de) IRPF (Imposto de Renda Pessoa Física), desenvolvido pela Secretaria da Receita Federal. O IRPF é uma descrição de como construir uma máquina de calcular impostos, a qual concretizada nas máquinas possuídas pelos contribuintes, desempenha tarefas peculiares ao sistema tributário brasileiro, de um modo compreensível por pessoas físicas que precisam realizar o ajuste anual de contribuições de impostos frente à Fazenda Nacional. Suponha que o usuário deseja usar tal máquina, e possui um sistema de computador (MPU) constituído por um PC + sistema operacional + browser web. Pode-se proceder conforme as etapas a seguir; 1– o usuário emprega MPU para carregar e interpretar o browser web, construindo MCSW1, que realiza a tarefa de copiar o (software) instalador do (software) IRPF, da máquina servidora da SRF, para MPU. Isto provoca uma alteração em MPU, que passa a ser MPU', agora constituída por um PC + sistema operacional + browser + instalador do IRPF;

2 – o usuário emprega MPU' para carregar e interpretar o instalador do IRPF, construindo MCSW1', a verbalização de MCSW1' instala o software do IRPF sobre MPU', que passa a ser MPU", composta por um PC + sistema operacional + browser + instalador do IRPF + IRPF. Todos esses usos de máquinas ainda não provocaram o efeito final desejado, que é o uso da máquina de calcular impostos.

3 – por fim, o usuário emprega MPU" para carregar e interpretar o (software) IRPF, o que constrói MCSW1", para realizar a atividade originalmente necessária.

Onde estão o software se as máquinas em cada um dos momentos descritos acima?

- Quatro máquinas no momento 1: (1) MCSW1, que foi construída por (2) MPU sob demanda do contribuinte. MCSW1 foi usuária de (3) máquina servidor da SRF, e construiu (4) MPU'.
- Três máquinas no momento 2: (1) MCSW1', que foi construída por (2) MPU' sob demanda do contribuinte. O resultado da verbalização de MCSW1 construiu MPU".
- Duas máquinas no momento 3: (1) MCSW", que foi construída por (2) MPU" sob demanda do contribuinte, para ser utilizada no cálculo de ajustes de contribuição de impostos.

Onde estão o usuário, o cliente e o desenvolvedor no cenário acima? O usuário atuou nos três instantes, usando diretamente seis máquinas (MPU, MCSW1, MPU', MCSW1', MPU" e MCSW"), em momentos distintos, e mais uma máquina (servidor SRF) através de intermediação. Embora a intenção do contribuinte fosse usar a máquina IRPF, ele precisou empregar seis outras máquinas distintas para usar a máquina desejada.

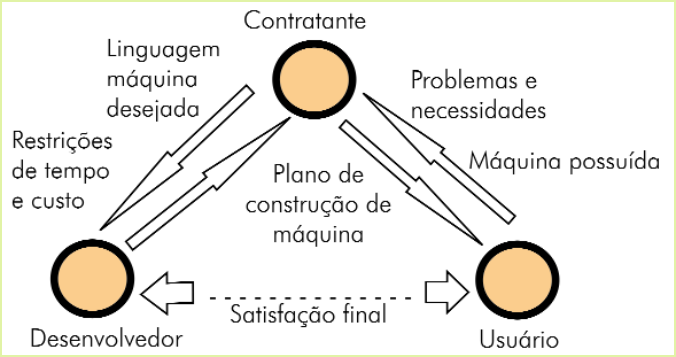

Figura 3 - Relação entre praticantes do software

O cliente do IRPF agiu antes dos momentos 1, 2 e 3, quando identificou uma série de problemas relativos aos custos e prazos de processamento de ajustes anuais de impostos de pessoa física. O cliente, nesse caso, é um agente (coletivo) do governo federal que descreveu um conjunto de formulários, tabelas de contribuição, mecanismos de cálculo, etc., que constituem o que se pode chamar de "linguagem de ajuste anual de constribuições da fazenda nacional baseada na plataforma de sistemas de computador PC", ou simplesmente a "linguagem do IRPF". Além da linguagem, o cliente definiu condições adicionais de prazo e custos para a construção do plano de máquina IRPF, que delimita a ação dos desenvolvedores. A definição da linguagem do IRPF não foi realizada em um momento único e isolado, dentre outras coisas porque a concepção de como serão solicitadas e realizadas as tarefas de uma máquina de calcular impostos (de qualquer tipo de máquina em geral) não possui uma solução única. Ela precisa ser projetada. No caso específico da "linguagem do IRPF", a mesma tende a sofrer pressões para evoluir rapidamente devido à natureza coletiva de seu uso, por milhões de usuários. Percebe-se portanto que, antes de ser imutável, a linguagem do IRPF, como a linguagem da maioria das máquinas desenvolvidas, é o resultado de um processo criativo que se desenvolve ao longo de vários anos, e que tende a evoluir da mesma forma que evoluem as linguagens de manipulação de máquinas físicas como controles remotos, liquidificadores, relógios e painéis de automóveis. Finalmente, por estar sujeita a um intenso processo de reprodução, adaptação, mutação e seleção, causada por um conjunto de tentativas e erros para criar a linguagem mais compreensível e verbalizável por um conjunto aberto de seres cognitivamente ativos, as linguagens das máquinas computáveis evoluem de forma darwiniana, sendo artefatos *designoids* (2), tais como o são as enzimas, glândulas, sistemas neuro-musculares, organismos e ecossistemas.
O desenvolvedor do IRPF atuou baseado nas seguintes restrições determinadas inicialmente pelo cliente: conceber um plano de construção de uma máquina capaz de verbalizar a "linguagem do IRPF"; conceber um plano de construção de máquina que seja carregável e interpretável em cada uma das máquinas possuídas por cada um dos usuários do software; conceber um plano de construção de máquina que seja concluído dentro das restrições de prazo e custos determinados.
Estabelecida essa tríade dos envolvidos na prática do software, é possível realizar uma série de análises de cenários e variações da qual emerge a riqueza e possibilidades do software em nossa sociedade.

O cenário percebido pelo usuário

A complexidade das máquinas possuídas - MPUs Um dos problemas que mais complicam a relação entre o usuário e a máquina construída pelo software, é a grande complexidade das MPUs sobre as quais o software é interpretado. Através do exemplo do IRPF, percebe-se que a MPU é constantemente compelida e sujeita a modificações, em especial pela própria ação do usuário. Eventualmente, cada uma das ações realizadas sobre as MPUs diretamente manipuladas pelo usuário implica em alterações em dezenas ou mesmo centenas de outras máquinas hierarquicamente equivalentes ou inferiores, que são postas em atividade e inatividade no momento em que se está usando um sistema de computador. Essa complexidade e constante mutação provocam o surgimento constante de diferenças entre a MPU que havia no momento em que o cliente identificou uma solução, e a MPU do usuário no momento em que o software é carregado e interpretado. A solução prática para esse problema surge através da necessária padronização de configurações ou plataformas de sistemas de computador, que consiste em estabelecer uma máquina padronizada, capaz de executar tarefas úteis a uma grande quantidade de planos de construção de máquinas. Ao se criar um mínimo denominador comum de máquina possuída pelo usuário, é possível criar planos de construção capazes de serem interpretados em uma grande quantidade de sistemas de computador. Termos e conceitos como Pentium, IBM PC, Macintosh, Linux, Unix, MS Windows, Máquina Virtual Java, MS Office e .NET denotam plataformas de sistemas de computador de diversos níveis hierárquicos, sobre as quais é possível executar categorias específicas de software. Uma das mais importantes práticas do software é identificar qual é a plataforma de máquina que apresenta as melhores vantagens no contexto específico de uma relação entre usuários, clientes e desenvolvedores.

A evolução das linguagens necessárias - LNUs Máquinas são extensões do ser humano. São mídias através das quais se estabelecem comunicações com resultados úteis e previsíveis. A natureza das linguagens de comunicação usuário-máquina permeia profundamente toda a relação e história do homem e dos artefatos que constrói, possuindo um impacto profundo sobre as atividades produtivas da sociedade. Criar linguagens está, portanto, no cerne da ação humana, e a prática do software permite o exercício desse processo criativo de forma eficiente e reproduzível (nos milhões de sistemas de computador que existem) como jamais se viu na história da humanidade. A definição da linguagem verbalizada por uma máquina computável é um processo criativo e evolutivo, baseado em experiências cognitivo-coletivas. Espera-se portanto que, da relação usuário-cliente-fornecedor, seja possível a definição de uma linguagem necessária (LNU) para solução do problema que o usuário tem em mãos. Tal solução é dificilmente obtida de forma plenamente satisfatória, a menos que o usuário seja outra máquina, devido aos seguintes fatores:
1- do mesmo modo que a interação (conversação) é essencial ao aprendizado de um novo idioma ou à manipulação de uma máquina como um automóvel, a compreensão de uma LNU (e das capacidades e limitações da máquina que a verbaliza) só atinge a plenitude no limite da interação entre o usuário e uma máquina que se aproxime da máquina necessária (MNU). Desse modo, a concepção de LNUs satisfatórias é um processo interativo, que envolve a definição de várias linguagens necessárias intermediárias (LNI', LNI", etc), que são verbalizadas por MCS, que são construídas por planos de máquinas intermediárias (SWIs). A única forma de conferir satisfação plena e duradoura ao usuário é quando o mesmo é também uma máquina que não evolui sua capacidade de verbalização, como ocorre com muitas máquinas mecânicas. Neste caso, a LNU pode ser plenamente definida através de uma gramática e lógica matematicamente formais. Esta última situação caracteriza o desenvolvimento do que Lehman (2) chama de S-Type Program, que é um software cuja corretude (satisfatibilidade) é plenamente definida a partir de sua (especificação de) linguagem.
2- o segundo fator que dificulta a satisfação plena do usuário é que o processo interativo está baseado em relações humanas. Ao interagir com quaisquer das máquinas intermediárias o usuário adquire compreensão prática de como a máquina verbalizou a sua comunicação, fornecendo soluções (de algum modo satisfatórias) para os problemas e necessidades com as quais o usuário se depara. Dado que as diversas soluções intermediárias são produzidas ao longo do tempo, alguns problemas originalmente percebidos pelo usuário se mostram solucionados, novos problemas surgem à medida em que o

ambiente do usuário se transforma pela ação da máquina, e problemas que não eram percebidos se tornam aparentes ou desaparecem. Essas transformações emergem dentro de um contexto de negociação de prazos e custos entre o cliente e o desenvolvedor, onde cada um dos agentes procura otimizar a relação custo-benefício do seu ponto de vista. Os cenários acima contribuem para que a LNUs sejam sujeitas a um forte processo de seleção, adaptação e mutação (devido a inevitáveis erros no processo de comunicação entre as partes), caracterizando o que Lehman(2) chama de E-Type Software, ou *real-life* software, cujo desenvolvimento se processa através de múltiplos níveis, múltiplos agentes e múltiplos ciclos de feedback positivo e negativo.

A ORGANIZAÇÃO DAS MÁQUINAS NECESSÁRIAS O maior diferencial qualitativo do computador, relativo a todas as outras máquinas criadas pelo homem, é a capacidade de manipulação de representações simbólicas e discretas, estruturadas na forma de linguagens computáveis. Uma linguagem computável, verbalizada por uma máquina de ordem 0, é capaz de, ao interpretar um software nessa linguagem computável, criar uma máquina de ordem 1, cuja linguagem verbalizada apresenta o mesma expressividade da linguagem da máquina de ordem 0. Em outras palavras, embora cada linguagem computável apresente características peculiares de sintaxe e semântica, é possível ao desenvolvedor do software que interagirá com uma máquina computável, conceber uma máquina de maior ordem, que terá o mesmo poder computacional da máquina original. A esse conceito fundamental de computabilidade soma-se a capacidade que os sistemas de computador tem de: verbalizar a linguagem em alta velocidade, em conformidade com planos de máquinas; e produzir resultados coerentes com interações imprevisíveis efetuadas pelos usuários, sobre os quais os sistemas não exercem controle.

...O DESIGN VERBALIZA A LINGUAGEM DEFINIDA PELO CLIENTE...

A computabilidade confere elevada capacidade e flexibilidade de ação do desenvolvedor na organização do software em vários níveis de hierarquia e interdependência, enquanto a interatividade e imprevisibilidade do usuário produzem o indeterminismo nos resultados das máquinas sobre os sistemas com os quais interagem. Esta combinação entre poder teórico e imponderabilidade prática cria uma condição para o surgimento de problemas de complexidade, falhas de planejamento e baixo grau de satisfação, que se espera possam ser controlados através de práticas da engenharia de software.

A ENGENHARIA DE SOFTWARE É a disciplina do conhecimento humano que tem por objetivo definir e exercitar processos (humanos atuando como máquinas), métodos (planos de processos), ferramentas e ambientes (máquinas apoiando processos e métodos) para construção de software que satisfaça necessidades de cliente e usuário dentro de prazos e custos previsíveis. Em outras palavras, engenharia de software é uma atividade industrial de produção de software, através da qual são produzidos vários artefatos não necessariamente compreensíveis por máquinas, mas que contribuem decisivamente para que um plano de construção de máquina seja satisfatoriamente criado. O corpo de conhecimentos da engenharia de software é estruturado em áreas (3). Uma interpretação de cada área é fornecida abaixo, relacionando-as com os conceitos previamente estabelecidos na relação cliente-usuário-desenvolvedor.

1. Requisitos de software – atividades para aumentar a precisão e controlar a variação da linguagem de interação entre a máquina desejada e o usuário.
2. Design de software – atividades para particionar e organizar internamente o plano de construção global do software, subdividindo-o recursivamente em diversos planos de máquinas (e suas linguagens) internas, até que cada plano de máquina seja realizável custo-efetivamente com o auxílio da MPU ou com outras internamente planejadas.
3. Construção de software – atividades para definir instruções e descrições para cada um dos planos de máquina, de modo que cada plano de máquina seja carregável e interpretável pela MPU ou por uma das máquinas internamente construídas.
4. Testes e qualidade de software – atividades para atestar que: o software será adequadamente interpretado nas máquinas dos desenvolvedores e dos usuários; a construção de software foi feita conforme os planos do *design*; que o design verbaliza a linguagem definida pelo cliente; e o usuário terá suas necessidades satisfeitas através do software.
5. Manutenção de software – atividades tipicamente aplicadas ao desenvolvimento de E-Type Software, para permitir que, ao longo dos vários ciclos de interação de re-definições de linguagens da máquina necessária (mudanças de requisitos) sejam tomadas ações para tratar o efeito das leis de evolução do software estabelecidas por Lehman. Essas leis são principalmente: (I) a Lei da Mudança Contínua – deve-se permitir que sejam feitas alterações no plano de software para verbalizar a linguagem modificada; (II) Lei da Complexidade Crescente – devem ser empreendidos esforços para tornar sob controle a complexidade que cresce à medida que alterações são conduzidas no software; (III) Lei da Auto-Regulação e (IV) Lei da Conservação da Estabilidade Organizacional – que sejam mantidas a estabilidade dos atributos dos processos e do produto (software) frente à estrutura da organização produtora de software, de modo que organização mantenha uma taxa líquida de produtividade, invariável ao longo do ciclo de vida do software; (V) Lei da Conservação da Familiaridade – que a taxa de mudanças nos sucessivos *releases* do software se mantenha constante ao longo do tempo; (VI) do Lei Crescimento Contínuo – que a quantidade de interações ou tarefas verbalizadas pelo software cresça para que se mantenha o nível satisfação do usuário; e (VII) Lei da Qualidade em Declínio – que constantes mudanças na máquina possuída pelo usuário devem ser feitas sob controle rigoroso da manutenção e adaptações constantes ao software, caso contrário a qualidade do mesmo declinará perante o usuário.
6. Gerência de configuração – atividades para garantir que seja adequadamente disponíveis para uso pelos desenvolvedores de software a estrutura interdependente-hierárquica de máquinas: 1- possuídas pelo usuário; 2- internamente construídas através do software; e 3- que apóiam os processos de engenharia de software.
7. Gerência de engenharia – atividades para garantir que o cliente receberá o software em conformidade com a LCU e outras restrições combinadas entre as partes.
8. Processos de engenharia – atividades para planejar, suportar, monitorar, controlar e ajustar todas as outras áreas e atividades de desenvolvimento, de modo a estruturá-las adequadamente na forma de processos produtivos reproduzíveis e previsíveis.
9. Ferramentas e métodos – atividades de seleção e adoção de máquinas e

planos de processos que aumentem a produtividade dos desenvolvedores enquanto reduzem a ocorrência de falhas no desenvolvimento.
Até este momento a definição de software usada se refere à noção de plano de construção de máquina. Será esta a definição mais aceita para software entre os praticantes? Vejamos abaixo outra definição de programa (software) presente em uma licença de software da IBM (4).
'... O termo "programa" significa o programa original e todas as cópias completas ou parciais do mesmo. Um programa consiste em instruções legíveis por máquina, seus componentes, dados, conteúdo audiovisual (tal como imagens, texto, gravações ou figuras) e materiais licenciados relacionados.'.
Conforme esta definição, em geral adotada pela indústria de software, um software é mais do que as instruções interpretáveis por uma máquina. Digno de nota é a indicação de que conteúdo audiovisual (tal como imagens, texto, gravações ou figuras) tambem é parte do software - este aspecto extrapola o conceito de software inicialmente apresentado como meta-máquina, à medida que torna explícito o fato de que qualquer material escrito, impresso, apresentável em qualquer mídia de comunicação, de natureza textual, gráfica, audível, etc, que tem por objetivo descrever algo para o usuário ou sua máquina, também é parte do software.
Outra definição de software comumente aceita entre quem desenvolve software é a que prescreve que o resultado de quaisquer das atividades do processo produtivo de software também é software. Além de todas as mídias digitais, impressas, ou reproduzíveis de alguma forma, que foram reproduzidas e entregues ao cliente, são parte do software os subprodutos internos do processo produtivo, como planos de decomposições de software, especificações de linguagens, definições de prazos e custos limites, planos de testes, documentos formais de aceitação, etc. Cada um dos artefatos é parte de um plano de construção, não necessariamente compreensível por uma máquina computável, mas destinado a ser interpretado por um ser humano que participa da construção e evolução do software. Sendo assim, uma possível eliminação desses artefatos (ou do conhecimento neles contidos) de dentro da composição do software, sempre provoca prejuízos no processo de manutenção do mesmo.

Conclusões A prática do software emerge da interação entre múltiplos agentes coletivos, com interesses e necessidades distintas, que contribuem com pontos de vista complementares para usar e criar máquinas, linguagens e planos de construção de máquinas. Embora a satisfação primária provocada pelo uso do software seja resultante do efeito imediato de uma relação mecânica de interpretação efetuada por uma máquina computável, o contexto histórico-social-lingüístico de concepção do software o redefine como um artefato modularizado, interdependente e hierarquizado, constituído por mídias de diversas naturezas, concebidas por uma ampla gama de seres humanos com habilidades profissionais extremamente variadas, e destinadas não só à interpretação por máquinas computáveis, mas também por seres humanos.

Jorge Henrique Cabral Fernandes *é professor do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte*

Referências bibliográficas

1 Dawkins, R. *Clibimg mount improbable*. W.W Norton, 1996.
2 Lehmam,M.M. *Laws of software evolution revisited*. LNCS 1149, Springer, 1997.
3 SWEBOK Project. *Software engineering body of knowledge*. Ed. 0.9. 2001.
4 IBM software license

Refinamento: a essência da engenharia de software

Ana Cavalcanti

A noção de refinamento captura a essência das atividades diárias de engenheiros de software, que projetam sistemas baseados em especificações, e de programadores, que implementam estes projetos. Em ambos os casos, o principal objetivo é a construção de sistemas e programas de acordo com documentos que os definem. O produto final, acima de tudo, deve ser, ou tem que ser, correto.
Refinamento é a relação que existe entre uma especificação, seus projetos e implementações corretas, do ponto de vista funcional. Métodos de desenvolvimento de programa são baseados nesta noção de uma forma ou de outra. Uma técnica formal vai além, no sentido que ela provê uma base matemática para garantia de correção. Neste caso, a meta primordial é o refinamento de uma especificação inicial para obtenção de uma implementação aceitável. Critérios de aceitação podem incluir, por exemplo, eficiência, mas a garantia fornecida é que a especificação e a implementação estão relacionadas por refinamento.

Conceitos Básicos Inicialmente, refinamento foi estudado para programas seqüenciais, aonde o foco é a relação entre as entradas e saídas de um programa. Foi identificado que há basicamente duas formas de refinar uma especificação. A primeira é a introdução e a transformação de estruturas de programação e controle como atribuições, condições, e laços. Isto é chamado refinamento algorítmico.
A segunda forma de refinamento é relacionada com as estruturas de dados usadas no programa. Sistemas são especificados em termos de tipos de dados que são apropriados para descrever propriedades do domínio de aplicação; neste estágio do desenvolvimento, não se faz, por exemplo, considerações relacionadas à eficiência.
Decisões de projeto, no entanto, normalmente introduzem estruturas de dados mais elaboradas e apropriadas para implementação. A mudança de representação de dados envolvida nessa tarefa é chamada refinamento de dados.
O ponto de partida de qualquer método formal é uma especificação formal. Correção é uma noção relativa: dizemos que um programa é correto se ele implementa a sua especificação. Para garantir correção, nós precisamos de uma especificação formal do programa.
Há muitas linguagens e formalismos em uso hoje. Nós usaremos uma linguagem de especificação chamada Z para apresentar um exemplo. Uma especificação de sistema em Z consiste basicamente de uma definição de um estado e de uma coleção de operações. O estado é composto de variáveis que representam os dados usados e registrados no sistema. As operações recebem entradas e produzem saídas, possivelmente alterando o estado.
Tanto o estado quanto as operações são definidas por esquemas: uma notação gráfica para agrupar declarações de variáveis e suas propriedades.

Exemplo Nós apresentamos a especificação de um sistema que calcula a média de uma seqüência de números recebidos como entrada. O estado deste sistema só tem um componente: a seqüência de inteiros.