

Monografia de Engenharia de Software

Ferramentas CASE: Suporte, Adoção e Integração

Adalberto Cajueiro de Farias

Universidade Federal de Pernambuco
Centro de Informática
Cx. Postal 7851– CEP 50.740-540 Recife (PE)
acf@cin.ufpe.br

Resumo: o crescente aumento da complexidade dos sistemas de Software fez surgir a necessidade da existência de uma sistemática para modelagem de problemas do mundo real em software. Ao final da década de 60, a Engenharia de Software deu seus primeiros passos no sentido de estabelecer essa sistemática para desenvolvimento de software. Atualmente, Engenharia de Software pode ser compreendida como o uso de técnicas, teorias, métodos e ferramentas necessárias para desenvolver software de qualidade. As ferramentas de apoio ao processo representam um grande passo no sentido de diminuir esforços humanos e garantir a corretude do produto. A presente monografia é referente à disciplina de Engenharia de Software do curso de Mestrado em Ciências da Computação do Centro de Informática da UFPE, ocorrida no primeiro semestre de 2001. Nela abordamos alguns temas relativos ao suporte de ferramentas CASE (*Computer Aided Software Engineering*) no processo de software: sua importância, benefícios, processo de adoção e problemas de integração. Esperamos com isso que este documento sirva como um guia introdutório para curiosos que desejam descobrir a importância das ferramentas CASE num processo de software e conhecer um pouco o estado da arte da integração entre as mesmas.

Palavras-chave: Engenharia de Software, CASE, processo, integração, qualidade, ambiente.

1. Introdução

Os sistemas de software desempenham um importante papel no mundo real. Em seus diferentes âmbitos de atuação, os produtos se mostram cada vez mais adequados às necessidades impostas pelos usuários. Seja por interesse industrial, comercial, militar ou científico, o objetivo final dos sistemas computacionais é sempre a qualidade.

Tratando o software como um produto, a Engenharia de Software vem desempenhando um papel muito importante no desenvolvimento de programas de computador. Desde a elicitação¹ aos testes, o processo² de software sofreu diversos aprimoramentos e hoje é compreendido como essencial para a criação de qualquer sistema computacional. Notações, formalismos, modelos e metodologias surgem para dar suporte ao processo, reduzindo os custos envolvidos, eliminando os problemas inerentes do desenvolvimento *ad hoc*, aumentando a produtividade dos desenvolvedores e melhorando a qualidade do produto final.

Como forma de incorporar rigor matemático ao processo, Métodos Formais têm desempenhado um papel muito importante no desenvolvimento de software. Notações apropriadas permitem eliminar ambigüidade. Técnicas de verificação garantem a corretude das propriedades dos sistemas antes mesmo de serem desenvolvidos. Regras de transformação permitem a especificação de sistemas em uma linguagem mais abstrata, de forma que uma

¹ Captura dos requisitos de um sistema para sua posterior modelagem e implementação.

² Processo de software é o conjunto de todas as atividades relacionadas ao desenvolvimento, controle, validação e manutenção de um software operacional [04].

especificação mais concreta seja semanticamente idêntica à abstrata podendo inclusive ser automaticamente derivada através do uso de ferramentas de apoio.

A sofisticação dos métodos de desenvolvimento acarreta o aumento da complexidade da administração do processo. Tal como uma fábrica, a produção de software demanda de ferramentas de apoio capazes de eliminar problemas usuais decorrentes das limitações humanas.

Com o aumento dessa demanda, surgiram inúmeras ferramentas CASE, fornecendo suporte as diversas fases do processo de software. Embora representem um diferencial no processo de software, os maiores benefícios do uso das ferramentas advêm com a integração. Por exemplo, no desenvolvimento de projetos de grande porte uma ferramenta (ou conjunto de ferramentas) é utilizada em cada fase do processo. Uma ferramenta de Análise estruturada é usada para definir processos e fluxos de dados, outra para desenvolver o modelo entidade-relacionamento. A correspondência entre os documentos é garantida pelos engenheiros, tornando-se vulnerável a erros humanos. Com sistemas integrados, tais problemas são automaticamente resolvidos pelas ferramentas, eliminando erros e diminuindo o trabalho dos desenvolvedores [20].

O presente trabalho está organizado como segue. Na seção 2 revisamos um conceito muito importante: Qualidade de Software. Na seção 3 falamos um pouco sobre Crise de Software, citando Métodos Formais como uma alternativa plausível para resolução/redução do problema. Nas demais seções, abordamos as ferramentas CASE e suas contribuições para o desenvolvimento de software, falamos a respeito dos modelos de processos de desenvolvimento, mostramos as classificações mais comuns de ferramentas, processo de adoção e finalmente tratamos das questões sobre integração entre ferramentas CASE. O Apêndice ao final contém o significado das siglas utilizadas neste documento.

2. Qualidade de Software

Uma primeira compreensão do conceito de software seria: um programa de computador, construído com o propósito de executar tarefas desejadas em um certo tempo. Software nada mais é do que uma lógica construída e posta em um computador para ser executado. Complexo ou não, todo programa tem uma forma de ser planejado, construído testado e mantido.

A qualidade de um software constitui-se das características mostradas pelo produto, uma vez instalado e colocado em uso [03]. Diversos atributos podem ser levados em consideração quando da medição da qualidade de um programa. Os atributos essenciais³ do software são:

- Manutenibilidade – facilidade em se modificar o software de modo a corrigir erros ou adequá-lo à mudança dos requisitos.
- Dependibilidade – não devem causar danos econômicos ou físicos na ocorrência de falhas. Engloba características como confiança e segurança.
- Eficiência – diz respeito à execução em tempo aceitável sem comprometer os recursos do computador, tais como memória e processador.
- Usabilidade – provimento de uma interface de usuário e documentação apropriados.

Maximizar um atributo pode significar aumento no custo de desenvolvimento ou o comprometimento de outro atributo. Por exemplo, aumentar a eficiência pode significar um aumento exponencial do custo [03].

³ São os principais atributos que devem ser levados em consideração quando da medida da qualidade de software. Outros atributos são: corretude, confiança, robustez, portabilidade, modularidade, flexibilidade, testabilidade, interoperabilidade etc.

3. Crise de Software

A Engenharia de Software procura desenvolver software a um custo baixo, de forma que o tempo seja um parâmetro controlável. A qualidade do produto final é função de seu processo de desenvolvimento. Melhorando o processo, o produto terá maior qualidade.

Certos sistemas de softwares podem ser julgados como produtos caros devido a sua manutenção, que custa em média 4 vezes mais que o desenvolvimento [03].

Manter sistemas significa modificá-los de alguma forma para que eles realizem as funcionalidades desejadas. Os tipos de manutenção podem ser:

- *Manutenção corretiva* – correção de erros detectados no software. Quanto mais abstrato for o erro, mais caro e difícil será sua correção: erros de código são geralmente fáceis e baratos de serem corrigidos, erros de design podem afetar diversos componentes do sistema, erros de requisitos podem requerer uma revisão de design do sistema.
- *Manutenção adaptativa* – modificar o software para alguma adequação à plataforma ou sistema operacional.
- *Manutenção de perfeição* – adequar o software aos novos requisitos do cliente ou às mudanças das regras de negócio.
- *Manutenção preventiva* – também conhecida como Reengenharia de Software. As mudanças são feitas no sentido de tornar o software mais fácil de ser corrigido, adaptado e melhorado para os usuários finais. Em resumo, este tipo de manutenção visa melhorar o software para aplicação dos três tipos de manutenção vistos acima.

Os custos de manutenção dependem de muitos fatores, sejam eles de natureza técnica ou não. Alguns desses fatores são: Independência Modular, Linguagem de Programação, Estilo de Programação, Validação e Teste, Qualidade da Documentação, Técnicas de Gerenciamento de Configuração, Estabilidade do Hardware etc.

O objetivo da Engenharia de Software é melhorar o processo de forma que a qualidade seja consideravelmente aumentada a um custo aceitável e o tempo de desenvolvimento não seja comprometido. Em [06] encontramos uma explicação realista e plausível da relação *Teoria x Experimento*. Computação não é uma exceção às demais Engenharias. As teorias falam a respeito da natureza da computação, o que é exequível e como pode ser gerenciado. A teoria aplicada à prática muda o foco do desenvolvimento de software de “arte” e “magia” para uma sistemática cujos métodos são matematicamente consolidados.

Embora não exista uma sistemática universalmente aceita para desenvolvimento de software, é fundamental o uso de notações, técnicas, métodos e ferramentas. Nesse contexto, Métodos Formais têm surgido como uma boa idéia para a produção de sistemas cuja natureza requer um forte embasamento matemático.

3.1. Métodos Formais e o desenvolvimento de software

Nos últimos anos, a abordagem formal vem sendo bastante empregada no processo de software. Os primeiros exemplos do uso de formalismos foram as linguagens de alto nível, criadas ao final da década de 50, cujas sintaxes passaram a ser definidas através de uma notação chamada *Backus Naur Form (BNF)*, uma notação matemática simples e precisa, usada para definir as regras de formação dos termos de uma linguagem. Linguagens descritas através de uma BNF podem ter sua ambigüidade eliminada mais facilmente do que as definidas usando linguagem natural.

Ao final de década de 60 surgiram os primeiros trabalhos usando matemática para descrever a semântica das linguagens de programação [07, 08], dando origem depois às Especificações Formais, textos expressos em uma linguagem baseada na matemática, usados para prover uma descrição não-ambígua do comportamento desejado de um programa, introduzindo possibilidade de prova de propriedades no mesmo [09]. Na metade dos anos 70, foi desenvolvido um sistema de raciocínio auxiliado por computador baseado no modelo

computacional de Danna Scott, juntamente com certas noções de estratégias e provas táticas, servindo como um provador de teoremas. Na década de 80, foi desenvolvido um sistema algébrico, conhecido por *Calculus of Communicating Systems*, capaz de expressar computação concorrente através do simples conceito de processo. Pouco a pouco, a tarefa de desenvolver programas passa a ser tratada como uma ciência e não como arte. Na busca por uma melhor qualidade, as pesquisas têm se concentrado muito em teorias matemáticas no sentido de buscar soluções para os problemas de software.

Sob o ponto de vista de suporte ao desenvolvimento, Métodos Formais são de extrema importância para assegurar a correção da especificação através da verificação. Embora seja um passo trabalhoso e caro, a verificação de especificações formais não precisa ser feita de forma clara para o usuário final. Ferramentas auxiliares ao processo podem prover essa transparência.

Procurando automatizar cada vez mais o processo de software, as ferramentas CASE devem, acima de tudo, ter seu desenvolvimento fortemente baseado em alguma técnica ou teoria formal, de modo a ter seu uso justificado. Embora a existência de diversas ferramentas CASE no mercado possa significar um grande aumento no suporte à melhoria do processo, é preciso ter bastante atenção ao adotar uma ferramenta de apoio ao desenvolvimento. Muitas delas, apesar de terem grande utilidade, possuem incompatibilidade de integração, representando um grave erro de escolha. Mais adiante, falamos sobre os principais problemas de integração entre diferentes ferramentas CASE.

4. Breve histórico sobre ferramentas CASE

Muitas pessoas empregam o uso da palavra “CASE” para designar ferramentas que auxiliam o processo de software. Na verdade o termo CASE denota uma tecnologia que consiste em fazer uso de ferramentas computacionais para o desenvolvimento de programas. É a Engenharia de Software combinada com ferramentas de apoio. Neste trabalho, referenciamos os softwares auxiliares ao processo através do termo “ferramentas CASE”, e a tecnologia em si através do termo “CASE”.

As primeiras ferramentas comerciais datam de 1982, no entanto só começaram a ganhar força no desenvolvimento de software a partir de 1985 quando a indústria desviou sua atenção para a produção de ferramentas cada vez mais sofisticadas. Em uma pesquisa realizada em 1990 [10], foram encontradas as seguintes evidências:

- Mais de 470 vendedores de ferramentas CASE foram identificados e novos deles aparecem a cada mês.
- O mercado de ferramentas CASE representou \$4,8 bilhões em 1990 e apresentou um crescimento estimado para \$12,11 bilhões em 1995.

Dois importantes avanços tornaram possível a sofisticação das ferramentas CASE:

- O rápido avanço do hardware em *workstations* e PCs, no início dos anos 80. O aumento de memória, da capacidade de processamento e o surgimento de uma interface gráfica facilitaram a interação *desenvolvedor x máquina*.
- As pesquisas na área de desenvolvimento de software deram origem a uma série de modelos que poderiam ser suportados por ferramentas.

Entre universidades, instituições governamentais e órgãos de defesa, os trabalhos de pesquisa e os investimentos têm aumentado consideravelmente no desenvolvimento de ferramentas CASE.

5. Importância das ferramentas CASE no processo de software

A seguir, damos uma visão geral dos principais modelos de processo de software. Cada um deles apresenta as atividades de maneira singular. Independente do processo de software escolhido por uma empresa, as atividades presentes no mesmo tendem a ser cada vez mais automatizadas por ferramentas CASE. Adotar um modelo padrão para o processo de software parece ser uma atividade muito difícil. A natureza do sistema a ser desenvolvido tem forte influência sobre o modelo a ser utilizado. Nem todas as atividades existentes em um modelo são adequadas para qualquer tipo de software. Entretanto, todos os modelos concordam em algumas atividades básicas que são:

- *Especificação* – definição das restrições, operações e funcionalidade do sistema.
- *Desenvolvimento* - Confeção do software a partir de seus requisitos levantados na especificação.
- *Validação* - comprovação do produto frente ao cliente para assegurar a correspondência do software com o que foi solicitado.
- *Evolução* – realização das mudanças de adaptação às novas necessidades do cliente.

Diferentes processos de software apresentam cada atividade do ciclo de vida de uma forma. Não faz sentido julgar um processo como “correto” ou “errado”. Mais coerente é afirmar qual processo é mais “adequado” para um determinado sistema ou organização. Por exemplo, no desenvolvimento de sistemas relacionados com Inteligência Artificial é mais adequado utilizar a abordagem RUDE (*Run, Understand, Debug, Edit*). A natureza de tais sistemas não se mostra clara o suficiente para que os requisitos sejam capturados e as funcionalidades representadas em algum documento.

Modelos de processos de software são ainda tema de pesquisa, no entanto está claro que, tal como as linguagens de programação, cada modelo de desenvolvimento de software tem seu paradigma [03]. A seguir damos uma rápida visão dos principais modelos existentes. Explicações detalhadas podem ser encontradas em [02, 03].

5.1. Modelo Cascata

Proposto em 1970, este modelo também é conhecido como Modelo Sequencial Linear ou Modelo Clássico do ciclo de vida do software. A Figura 5.1 mostra uma representação gráfica do modelo. As atividades são vistas como estágios do processo. Cada estágio serve para identificar problemas ocorridos no estágio anterior. Quando da detecção de algum problema o processo volta a sua fase anterior para correção. Na fase final, quando o software é colocado em uso, erros nos requisitos podem ser encontrados e corrigidos. Uma grande desvantagem dessa abordagem é que a detecção de um erro pode tanto causar um efeito de retorno das fases de desenvolvimento quanto causar novamente a execução das atividades (devido a um problema encontrado apenas na fase final do processo).

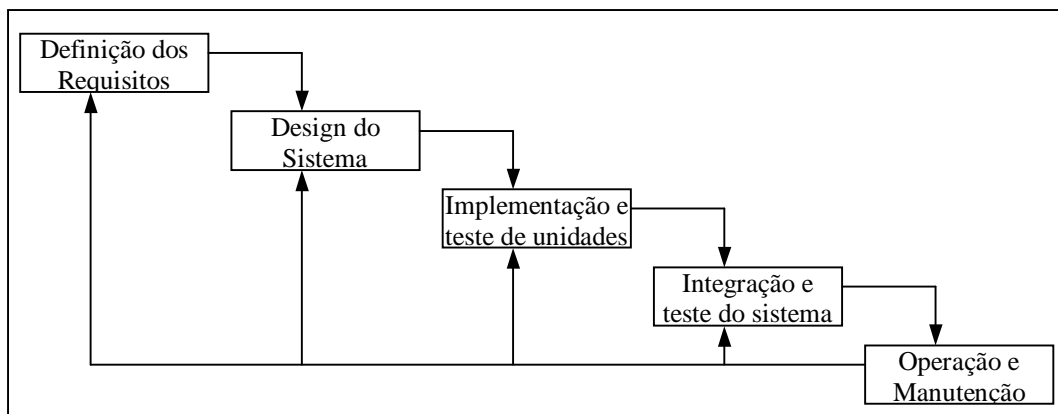


Figura 5.1: Modelo Cascata

5.2. Modelo de prototipagem

Modelo de processo adequado para os casos em que o cliente define os requisitos do sistema de forma geral, entretanto não detalha dados processados e saídas do sistema. Um protótipo é então desenvolvido e validado com o cliente, ajudando dessa forma a descobrir os próprios requisitos não fornecidos anteriormente. O protótipo serve apenas para dar uma primeira visão do sistema, em um tempo relativamente pequeno, porém pode causar sérios problemas devido aos seguintes fatores:

- O protótipo não representa versão alguma do sistema. Preocupado apenas com o produto final, o cliente passa a desviar sua atenção para o protótipo, esquecendo o sistema em si. O protótipo foge então do seu propósito para se tornar o produto principal.
- Quando do desenvolvimento do protótipo, os desenvolvedores não se preocupam em fazer os algoritmos da melhor forma, mas sim da forma mais rápida. Considerações importantes a respeito de plataforma e requisitos funcionais podem ser imprudentemente esquecidos, tornando o protótipo inadequado para ser tomado como base de um desenvolvimento.

5.3. Modelos de processos evolucionários

Em sistemas complexos, que requerem muito tempo de desenvolvimento, é natural que os requisitos mudem durante o processo. Os modelos evolucionários apresentam flexibilidade de adaptação a essas mudanças, sem comprometer o ritmo das atividades. Primeiramente uma linha principal de desenvolvimento é estabelecida e seguida. À medida que alguma adaptação é necessária, as mudanças podem ser trabalhadas numa outra versão do produto sem comprometer a linha principal de desenvolvimento. A cada mudança ocorrida durante o processo, versões mais completas do software são produzidas. A seguir, descrevemos rapidamente os modelos evolucionários utilizados na prática da Engenharia de Software.

5.3.1. Modelo Incremental

Modelo de processo que combina elementos do Modelo Sequencial com a filosofia iterativa da Prototipagem. Uma versão *core*⁴ do produto é estabelecida e desenvolvida. A partir de então os acréscimos e as modificações são feitos por etapas até a confecção do produto final. Este modelo torna possível a avaliação da primeira versão por parte do usuário e se adequa melhor aos projetos com *deadlines* muito curtos.

5.3.2. Modelo Espiral

Proposto por B. Boehm em 1988, este modelo faz uso da natureza iterativa da Prototipagem com os aspectos de controle e sistemática do Modelo Sequencial [02]. O processo é composto de uma série de ciclos compostos por todas as fases do processo. Ao término de um ciclo, as atividades são novamente realizadas num novo ciclo, representando o desenvolvimento de uma versão mais completa do produto. As atividades presentes em cada ciclo são:

- *Conversa com o cliente* – necessária para estabelecer uma comunicação efetiva entre desenvolvedor e cliente.
- *Planejamento* – definição de recursos, prazos e outras informações relacionadas ao projeto.
- *Análise de riscos* – avaliação dos riscos técnicos e gerenciais.
- *Engenharia* – construir uma ou mais representações da aplicação.
- *Construção* – construir, testar, instalar e prover suporte a usuário.

⁴ Uma versão contendo os requisitos básicos do produto.

- *Avaliação do cliente* – necessária para obter o *feedback* do cliente sobre o trabalho concluído até então.

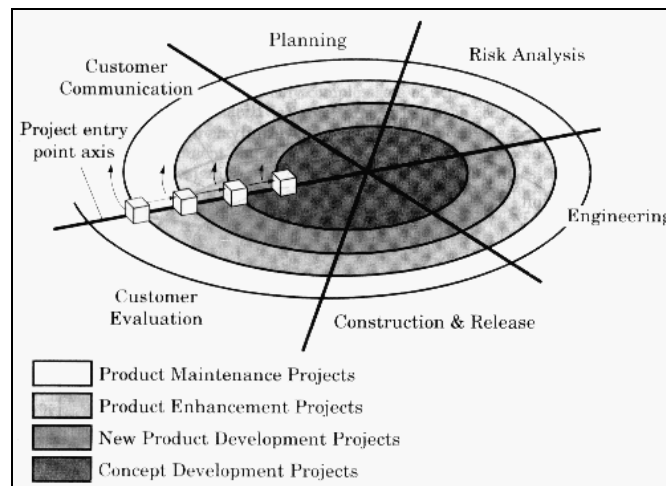


Figura 5.2: Modelo Espiral

Os ciclos são realizados no sentido horário e do centro para a extremidade (Figura 5.2). No primeiro ciclo, desenvolve-se a especificação, no segundo, um protótipo. Nos seguintes, versões mais detalhadas do produto podem ser desenvolvidas até que se atinja um estado final do produto e se inicie o ciclo correspondente à manutenção.

O Modelo Espiral representa uma abordagem bastante realística e adequada para desenvolvimento de sistemas em larga escala. A periodicidade das atividades pode ter muito impacto na qualidade do software: as interações com o cliente representam uma validação do que foi desenvolvido em cada nível (ciclo), a análise de riscos reduz problemas surgidos durante a evolução e a abordagem da prototipagem pode ser aplicada em qualquer estágio da evolução do produto. O modelo é relativamente recente e menos usado que as abordagens Sequencial e Prototipagem.

5.3.3. Modelo de Montagem de Componentes

Muito parecido com Arquitetura de Software [11], este paradigma incorpora muitas características do Modelo Espiral. É evolucionário no sentido de que demanda uma abordagem iterativa da criação do software, no entanto este modelo procura construir aplicações a partir de componentes de software previamente empacotados.

Estes “componentes”, às vezes chamados de “classes”, encapsulam tanto dados quanto as funcionalidades sobre estes dados. Se modelados e construídos corretamente tornam-se “peças” reutilizáveis em qualquer sistema baseado em algum modelo arquitetural⁵.

⁵ A Arquitetura de Software procura fornecer uma descrição do sistema a partir de seus componentes. Uma idéia principal da abordagem é poder identificar previamente os componentes e reutilizar algum já existente. Consiste mais em uma especificação do sistema em termos de componentes e interações entre os mesmos. É como se fosse uma organização composicional do software, uma representação ainda mais abstrata do que a especificação do sistema (modelagem de classes, casos de uso etc). Diversas definições de Arquitetura de Software podem ser encontradas em [11].

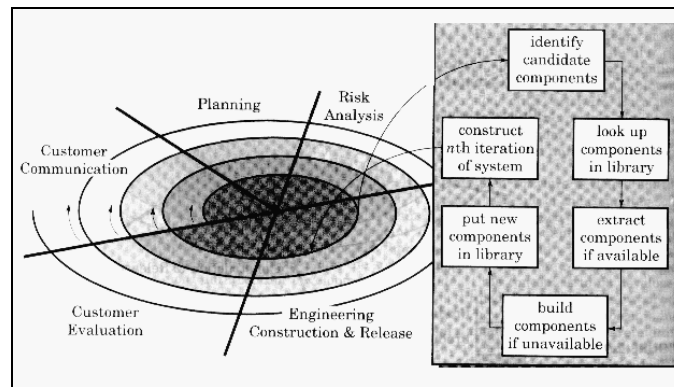


Figura 5.3: Modelo de Montagem de Componentes

De acordo com a Figura 5.3, as atividades de Engenharia e Construção do Software tornam-se uma única tarefa e sua forma de execução muda para, ao invés de pensar no que desenvolver, o objetivo é identificar quais componentes estarão presentes, quais deles existem (podem ser reutilizados) e quais precisam ser desenvolvidos. Desenvolvem-se os novos componentes empacotando-os e usando-os em seguida. A cada ciclo, novos componentes podem ser identificados e possivelmente desenvolvidos até que se atinja uma versão final do software.

O grande benefício deste modelo é o reuso de softwares existentes. Isso pode representar, segundo estudos da *QSM Associates*⁶, 70% de redução do ciclo de vida do software e 84% de redução dos custos de projeto, além de aumentar o índice de produtividade.

5.3.4. Modelo de Desenvolvimento Concorrente

Os modelos vistos até aqui são compostos por atividades realizadas por muitas pessoas dentro de uma organização. Na prática, as tarefas não são executadas de forma disjunta. À medida que os principais requisitos são capturados, dá-se início a fase seguinte visando o término de uma primeira versão. Sob este ponto de vista, muitas fases caminham em paralelo e as pessoas se esquecem da simultaneidade dessas atividades. As modificações nos requisitos devem ser refletidas para todas as atividades executadas, que dizem respeito ao requisito modificado.

Para tornar possível a propagação de uma mudança a diferentes executores, o modelo propõe que cada atividade passe a incorporar uma tarefa de análise de mudanças, executada como intuito de acionar diferentes partes ao mesmo tempo, caso um evento de modificação ocorra. O processo de software passa agora a ser representado por estados, cujas transições correspondem ao curso normal ou alguma correção a ser feita. A Figura 5.4 mostra este tarefa de análise para a atividade de Engenharia, entretanto cada atividade pode incorporar esta representação de forma semelhante.

⁶ *Quantitative Software Management Associates* – empresa enfocada em três áreas do desenvolvimento de software: Estimções de Custo, Controle de Projeto e Melhoria de Produtividade baseada em *benchmarking*.

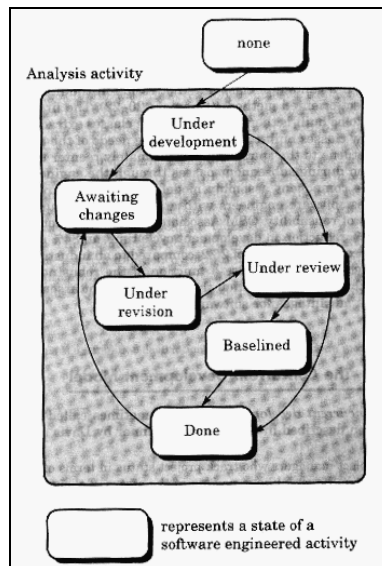


Figura 5.4: Modelo Concorrente

5.4. Modelos baseados em Métodos Formais

São modelos baseados em técnicas e notações matemáticas que reduzem bastante a ambigüidade, inconsistência e incompletude, comumente encontrados no desenvolvimento *ad hoc*. Entretanto, algumas razões tornam este tipo de paradigma de processo difícil de ser usado:

- Exigência de base teórica e matemática, coisa que poucos desenvolvedores possuem.
- Grande consumo de tempo e alto custo na qualificação de pessoal.
- Por ser um modelo baseado em métodos matemáticos, a validação normalmente é rejeitada pelo cliente por parecer difícil e complexa.

5.5. Técnicas de Quarta Geração

Oferecer um nível mais alto de abstração na modelagem e gerar código fonte a partir de modelos mais abstratos é o principal objetivo das Técnicas de Quarta Geração (4GT). O paradigma envolve a especificação em notações de alto nível ou notações gráficas. As ferramentas se encarregam de derivar especificações mais concretas a partir das mais abstratas.

Juntamente com as ferramentas CASE e geradores de código, a abordagem tornou-se mais geral na última década tem mostrado soluções para diversos problemas de software. É certo que, como todos os paradigmas, as Técnicas de Quarta Geração apresentam vantagens e desvantagens. O tempo de projeto reduz bastante e a produtividade das pessoas envolvidas na construção do software aumenta consideravelmente. Contudo, o código fonte produzido pelas ferramentas não é dos mais eficientes e a manutenibilidade de sistemas desenvolvidos com este paradigma ainda é uma questão em aberto [02].

Independente do paradigma de desenvolvimento de software utilizado, a produtividade nos processos aumenta significativamente quando as habilidades humanas são fortalecidas por ferramentas poderosas. “Um homem com um trator pode mover mais terra em um dia do que 50 homens trabalhando com ferramentas manuais” [03]. Com o propósito de aumentar a qualidade do produto e tornar o processo mais eficiente, as ferramentas CASE têm se tornado cada vez mais comum na indústria de software.

A tecnologia CASE pode ser tão simples quanto uma ferramenta singular que suporte uma atividade específica do processo, ou tão complexa quanto um ambiente que inclui ferramentas, pessoas, hardware, rede, sistemas operacionais, padrões etc [02].

A distância entre os requisitos e o produto final, considerando a produção em larga escala e a complexidade dos sistemas, vem fazendo das ferramentas CASE uma boa alternativa para o

problema da Crise de Software. Ferramentas auxiliares ao desenvolvimento contribuem (e sempre contribuirão) para o aprimoramento das técnicas da Engenharia de Software.

6. Categorias e classificação das ferramentas CASE

A tecnologia CASE pode ser dividida em três níveis:

1. *Tecnologia de suporte ao processo de produção* – ferramentas que suportam atividades do processo tais como especificação, design, implementação e teste.
2. *Tecnologia de Gerenciamento de Processo* - ferramentas de suporte à modelagem e gerenciamento do processo. Interagem com as ferramentas de suporte ao processo de produção com o intuito de direcionar e gerenciar suas tarefas numa atividade específica. São ferramentas ainda carentes de maturidade, sendo alvo de muita pesquisa.
3. *Tecnologia Meta-CASE* – ferramentas usadas para criar as ferramentas de suporte ao Gerenciamento de Processo e Processo de Produção. Algumas delas já foram desenvolvidas, porém não têm sido largamente adotadas pela indústria por serem difíceis de usar.

Existem diversos critérios para classificação de ferramentas CASE . Neste trabalho levamos em consideração a classificação por funcionalidade e por fase do processo de software.

6.1. Classificação por funcionalidade

A Tabela 6.1 mostra a classificação segundo a funcionalidade das ferramentas .

Tipo	Descrição	Exemplos
Gerenciamento	Servem para representar os elementos de um processo e seus relacionamentos. Úteis para entender o processo e atividades necessárias a sua execução.	Ferramentas de estimação de custos.
Edição	Úteis para codificação, modelagem gráfica e documentação.	Editores de textos, editores de diagramas .
Gerenciamento de Configuração	Ajudam nas tarefas de identificação, controle de versão, controle de mudança, auditoria etc.	Sistemas de controle de versão e sistemas de gerenciamento de mudanças.
Prototipagem	Ajudam na criação rápida de protótipos. Suporte a definição de <i>layouts</i> de telas, integração de telas com o <i>design</i> de dados, geração de código fonte.	Geradores de interfaces gráficas, linguagens de alto nível.
Suporte a programação	Permitem o desenvolvedor construir o software em uma linguagem mais inteligível para humanos.	Compiladores, interpretadores, editores.
Análise de programas	Analizam tanto o código fonte quanto interação com o sistema em execução. Úteis para gerar casos de testes e inspecionar o fluxo do programa.	Analísadores estáticos e dinâmicos.
Teste	Usadas para controlar e coordenar os testes de software.	Geradores de dados para teste, comparadores de arquivos.
Depuração	Servem para inspecionar o código fonte em execução, tornando possível uma observação rigorosa da execução do sistema.	Sistemas interativos de depuração.
Documentação	Ajudam na produção e publicação dos documentos.	Editores de imagens, processadores de textos.
Reengenharia	Servem para engenharia reversa, reestruturação de código.	Sistemas de reestruturação de programas

Tabela 6.1: Classificação por funcionalidade

6.2. Classificação por fase do processo

As ferramentas podem ser classificadas de acordo com as fases do processo nas quais são empregadas. A Tabela 6.2 lista as diferentes fases e mostra o uso das ferramentas em cada uma delas. O sombreamento indica a presença da ferramenta na fase específica.

	Especificação	Design	Implementação	Verificação/Validação
Geração de dados de testes				
Modelagem e simulação				
Transformação de programas				
Depuração interativa				
Análise de programas				
Processamento de linguagem				
Suporte a método				
Gerenciamento de interface de usuário				
Dicionário de dados				
Edição de diagramas				
Prototipagem				
Gerenciamento de configuração				
Preparação de documentos				
Edição de texto				
Estimação e planejamento				

Tabela 6.2: Classificação por atividade do processo

Apesar de estarem presentes em todas as fases do processo, as ferramentas CASE apresentam certa diferenciação em relação a qualidade do suporte provided. A Figura 6.1 mostra que certas atividades possuem ótima qualidade de suporte, enquanto em outras, o suporte é ainda precário, comprovando que estas atividades são pouco entendidas. Com a maturidade das ferramentas o suporte deficiente haverá de melhorar.

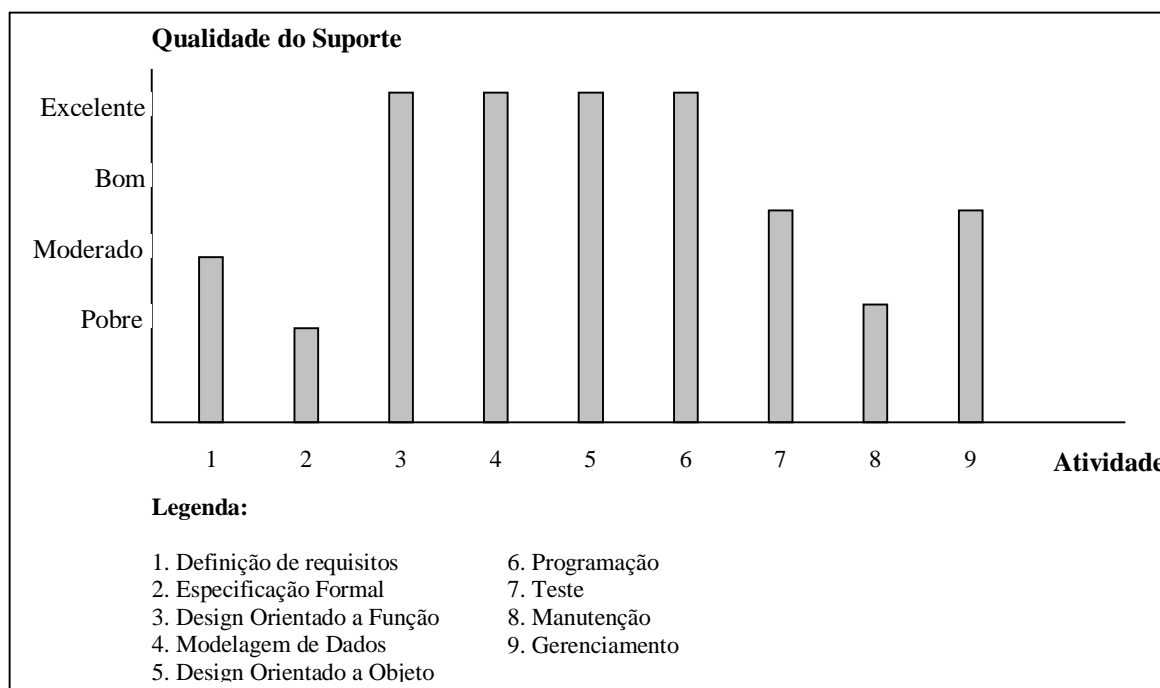


Figura 6.1: Qualidade do suporte CASE as atividades do processo

7. Processo de seleção e adoção de ferramentas CASE.

Nesta seção procuramos descrever alguns fatores a serem levados em consideração na adoção de uma ferramenta CASE. O processo de adoção depende muito do tipo de empresa, suporte desejado, custo e benefício esperado. Os tópicos são baseados no Workshop de Gerenciamento CASE, promovido pelo SEI (*Software Engineering Institute*), em Junho de 1991. Os diferentes temas abordados no *workshop* foram organizados nas seguintes seções⁷:

- Realidade das ferramentas CASE: o que elas realmente fazem – o que elas não fazem.
- CASE e Métricas
- Seleção de ferramenta CASE

7.1. Realidade das ferramentas CASE

Ferramentas CASE, apesar de contribuírem muito no processo de software, não resolvem todos os problemas. É preciso saber sua real contribuição, possibilidades e capacidades. É essencial ter uma visão realística das capacidades das ferramentas CASE, de suas deficiências e fornecer uma projeção sobre o futuro dessas ferramentas.

O que as ferramentas CASE realmente fazem?

7.1.1. Padronização dos Produtos

A diversidade de ferramentas no mercado é enorme, no entanto muitas delas são relativamente inflexíveis. Geralmente são apropriadas a um tipo de processo ou alguma outra tecnologia do fabricante. Os principais obstáculos da flexibilidade das ferramentas são:

- A maioria das ferramentas de design e análise possui algum tipo de dependência com um método particular, que nem sempre é adequado para todas as organizações.
- Muitos vendedores garantem que suas ferramentas são capazes de gerar documentos compatíveis com os padrões mais utilizados no mercado, quando na verdade, o suporte não é completamente automático e totalmente compatível. Normalmente o que existe é um *template* contendo dados julgados relevantes, ou espaços a serem preenchidos pelas ferramentas.
- Os padrões suportados pela análise de código das ferramentas são freqüentemente baseados em alguma heurística bem conhecida para qualidade de código. Isso pode se tornar uma inflexibilidade para uma organização que trabalhe com um padrão de codificação não entendido pela ferramenta.

É esperado que as ferramentas CASE passem a aceitar cada vez mais os padrões requeridos pelos clientes. A integração entre diversas ferramentas tende a suprir essa deficiência mais facilmente até que se chegue na geração de documentação completamente automatizada.

7.1.2. Automação do Processo de Software

Devido a algumas inflexibilidades, o suporte das ferramentas CASE ainda se limita a um certo número de métodos. Isso é devido a duas razões:

- Imaturidade da disciplina de Engenharia de Software, ocasionando uma carência de modelos de processos para automação. A automação deveria ser menos restrita a um modelo específico, diminuindo a dependência e inflexibilidade das ferramentas.

⁷ Apenas as seções de interesse para a confecção deste documento foram selecionadas.

- Imaturidade das ferramentas e ambientes, suportando pouca integração e associadas a um processo de software específico.

Os problemas de integração entre ferramentas serão resolvidos com o consenso dos fabricantes em determinar modelos compatíveis com qualquer ferramenta a ser desenvolvida. É esperado que os modelos de processo sejam mais bem entendidos. *Frameworks* e novos ambientes terão uma maior conformância com o processo e serão mais adequados à forma de trabalho dos engenheiros.

7.1.3. Reengenharia, Engenharia Reversa e Suporte à Reestruturação

Atualmente as ferramentas CASE provêm um suporte limitado às atividades de Reengenharia, Engenharia Reversa e Reestruturação. Em pouco tempo as aplicações poderão ser manipuladas ao nível de design. Decisões de mais alto nível serão propagadas até o nível de implementação aumentando a conformidade entre as diferentes atividades e documentos. Infelizmente, nem todas as decisões de projeto poderão ser recuperadas e reconstruídas. Muitas delas são decisões dos engenheiros e podem não ter sido corretamente armazenadas. A manutenção dos sistemas ocorrerá num alto nível de abstração. Os engenheiros pouco se preocuparão diretamente com código fonte. A partir de uma especificação funcional, tanto o design quanto o código poderão ser criados e manipulados.

7.1.4. Interoperabilidade entre ferramentas

Algumas diferenças como modelos de processos, tipo de suporte e dependência de fabricante, levam a uma interoperabilidade parcial das ferramentas. A maioria das conexões apresenta-se de forma unidirecional⁸. As razões dessas limitações são muitas, mas incluem principalmente:

- Proliferação dos padrões de interconectividade – muitos padrões de integração disputam lugar no mercado e oferecem diferentes tipos de interação. Essas diferenças são inerentes de intenções de domínio de mercado por parte dos desenvolvedores. Diante desse contexto, é difícil (ou até mesmo impossível) para um desenvolvedor particular suportar todos os padrões dos “gigantes” do mercado.
- Pobreza de interoperabilidade entre ferramentas de diferentes desenvolvedores, decorrente da competitividade entre os padrões.
- Pobreza de interoperabilidade entre as fases do ciclo de vida. Ferramentas criadas para uma fase específica do processo geralmente provêm pouco suporte a ligações com as demais fases. Normalmente os engenheiros fazem o *pipe* entre ferramentas pertencentes a diferentes fases do ciclo de vida. Esta atividade pode requerer um esforço grande para desenvolver um *link* lógico entre as ferramentas, sem considerar o aumento da dificuldade de manutenção de consistência entre documentos de diferentes fases.
- Natureza proprietária das arquiteturas. Poucos desenvolvedores estão dispostos a disponibilizar detalhadamente a forma de trabalho de suas ferramentas. Embora as interfaces de acesso sejam providas, é muito difícil dissecar a ferramenta até um nível onde ela possa ser efetivamente integrada. Nem sempre as interfaces de acesso provêm suporte à integração desejada.

É natural que cada ferramenta possua sua forma de representar os dados. A concordância de trabalhar em repositórios compartilhados ou facilidade em enviar e receber mensagens apresenta-se como uma opção comprometidora para interação de diferentes ferramentas em um

⁸ As transformações ou ações acontecem apenas em um sentido. Por exemplo, alguns sistemas refletem suas modificações de design no código gerado, entretanto poucos mantêm a consistência do design quando o código fonte é modificado. É preciso fazer novamente o processo de Engenharia Reversa.

ambiente de integração. Isso permitirá que *links* bidirecionais sejam estabelecidos, ajudando assim na interação entre diferentes fases do processo. As ferramentas passarão a ser desenvolvidas com suporte a utilização em ambientes de integração, eliminando os eventuais problemas de interação com ferramentas de outros vendedores.

7.1.5. Geração automática de código

A atividade de automatizar a geração de código é ainda imatura devido às seguintes razões:

- Pouco entendimento de métodos e técnicas necessárias para capturar adequadamente e expressar o design em termos de código fonte. Isso acontece porque não existe uma linguagem adequada para expressar o design.
- Carência de controle de otimização de código. Normalmente, os códigos gerados por ferramentas não representam de forma correta as restrições impostas pelos sistemas.

Atualmente, a geração de código fonte restringe-se à criação de código a partir de algum modelo de design (geralmente UML) e o código gerado possui algumas restrições. Por exemplo, para linguagens orientadas a objetos, as classes modeladas em UML são mapeadas em classes na linguagem, no entanto os métodos são escritos apenas com suas assinaturas. O corpo do método, assim como pré e pós-condições, não é transformado em código fonte, devendo ser inseridos manualmente. Com o aprimoramento/amadurecimento das notações, métodos, técnicas e a melhor integração entre as fases do processo de software, espera-se que toda atividade seja feita em um nível mais abstrato, permitindo também descrever por completo as funcionalidades. A depuração e a otimização também poderão ser automáticas.

7.1.6. Coleção de dados e Comunicação

Muitas ferramentas armazenam os dados em alguma base proprietária ou em arquivos manipulados diretamente pelas mesmas. Ao invés disso, os dados deveriam ser guardados em bases de dados acessíveis por diversas ferramentas. Essa carência se deve à pobreza em identificar o estado e riscos do projeto, sem falar na possível queda de performance na manipulação dos dados. Com a melhoria da tecnologia dos Bancos de Dados e surgimentos de modelos de processos mais precisos na medição do estado e dos riscos, espera-se que os dados sejam armazenados num repositório comum que forneça diferentes visões para clientes distintos e suporte representação dos diversos aspectos do ciclo de vida do software.

O que as Ferramentas CASE nunca farão?

Embora as ferramentas CASE facilitem bastante o desenvolvimento de software, elas possuem certas limitações. Existem algumas atividades que nunca poderão ser automatizadas, por exemplo:

- Minimizar ou simplificar o rigor intelectual em especificar, projetar, implementar e manter a qualidade de software.
- Corrigir problemas organizacionais ou suprir alguma necessidade de algum processo.
- Medir e certificar toda a qualidade do processo ou produto tal como exigido pelo usuário.
- Resolver problemas que requerem a habilidade e conhecimento de engenheiros.
- Prover interoperabilidade entre plataformas, ferramentas e fases do processo sem suporte adicional.

7.2. CASE e Métricas

A atividade de desenvolver software está muito relacionada com os custos de produção. As técnicas de estimação de custo possuem certas limitações e o suporte CASE pode ajudar bastante na análise quantitativa do produto⁹, tendo assim seu custo justificado. Utilizado como um sistema de medição, o *PEI* (*Production Efficiency Index*) aplica-se genericamente a qualquer processo de produção. Para os processos de software existe uma pequena variação do PEI que é utilizada, o *SPEI*¹⁰ (*Software Production Efficiency Index*) é necessário para avaliar o impacto da tecnologia CASE na eficiência da produção, independentemente da complexidade do produto, experiência do pessoal envolvido, características do ambiente de desenvolvimento etc. Este método pode ser utilizado também para avaliar os seguintes aspectos:

- Impacto das ferramentas CASE no processo de produção de forma detalhada: design, codificação e teste.
- Impacto dos outros fatores no processo: treinamento, melhoria do processo, melhoria de hardware.
- Identificar os fatores que impactam na eficiência de produção, servindo de base para o diagnóstico da produtividade e novas tomadas de decisões na organização.
- Prover formas de medições em função do tempo, ou seja, estimar produtividade, custos e tempo na construção de software.

7.3. Selecionando uma ferramenta CASE

A atividade de selecionar ferramentas CASE requer definição de muitos parâmetros a serem observados/avaliados. Os diferentes processos de seleção existentes não representam modelos gerais. Alguns pontos levados em consideração podem não ser úteis para todos os tipos de organizações. Mostramos agora as principais necessidades de avaliação surgidas quando da seleção de uma ferramenta:

- Identificar o problema a ser resolvido. Tão cedo quanto possível, o problema deve ser definido antes da compra da ferramenta. No mundo do desenvolvimento de software, dois anos após a aquisição, 70% das ferramentas não são mais utilizadas. Das que permanecem em uso, apenas 10% são usadas de forma correta. O problema pode ainda sofrer as seguintes influências: tipo de modelo de desenvolvimento, tarefas específicas, níveis do suporte, custo-benefício, riscos.
- Identificar os tipos de ferramentas disponíveis. Existem diversas listas de ferramentas CASE disponíveis no mercado. Algumas características merecem destaque: ano de fabricação, propósito, suporte à integração, preço, fabricante.
- Decisão em seguir algum processo de seleção. Normalmente, os modelos de processo de seleção são bastante variados e sempre possuem alguma característica divergente de algum objetivo/necessidade da organização. É comum tomar critérios previstos nos modelos como base para a escolha, no entanto adequações são inevitáveis, devido à complexidade/natureza de cada organização.
- Avaliar as ferramentas candidatas nos seguintes itens: facilidade de uso, poder, robustez, funcionalidade, facilidade de introdução na organização, qualidade do suporte.

O processo de adoção da tecnologia CASE é semelhante ao processo de desenvolvimento de software. A definição dos requisitos é essencial. Erros encontrados nesta fase são relativamente mais fáceis de se consertar. Quanto mais tempo demorar a descobrir um erro, mais cara será sua correção. A tecnologia CASE pode requerer muito esforço para adoção e implantação, mas

⁹ O suporte CASE para estimação de software apresenta-se ainda imaturo. As principais técnicas precisam se consolidar mais no mercado.

¹⁰ Maiores detalhes podem ser encontrados em [12].

podem ter seu custo justificado pelo benefício advindo do seu uso. Antes de tudo a organização deve ter um bom controle sobre o processo.

8. Processo de adaptação

Existem dois aspectos de adaptação que precisam ser discutidos. O primeiro diz respeito às ferramentas individuais, o segundo trata dos serviços integrados.

As técnicas mais comuns para as ferramentas individuais se aplicam a alguma forma de parametrização da ferramenta. Esta parametrização pode acontecer em vários tempos, com diferentes implicações na adaptabilidade. Exemplos de parametrização incluem:

- *Parametrização de tempo de instalação.*
- *Parametrização de tempo de execução compartilhado* – mais significativa para o processo que a anterior. Grupos de usuários utilizam diferentes ferramentas para as atividades do processo.
- *Parametrização de tempo de execução individual* – relacionada com o comportamento individual da ferramenta.

Estas formas de parametrização refletem os mecanismos para o processo de adaptação e devem ser consideradas como parte do mecanismo de integração. Algumas tarefas do processo suportam o uso direto de ferramentas CASE, enquanto outras dependem primordialmente da experiência do desenvolvedor e das convenções de uso.

Em termos de serviços integrados, as técnicas do processo de adaptação aplicadas aos serviços CASE integrados são mais difíceis de quantificar. Integração pode significar um relacionamento complexo. Nesse caso é de se esperar que o processo de adaptação seja mais lento em relação a uma ferramenta individual. Apenas com a evolução e melhoria da integração entre as ferramentas, as métricas quantitativas de adaptação podem ser precisamente definidas.

9. Blocos de construção e opções de integração

Conforme visto em [02], o desenvolvimento de ferramentas CASE deve seguir uma estrutura baseada em camadas (ou blocos), onde cada camada serve de base para a camada superior. A Figura 9.1 mostra a estrutura proposta.



Figura 9.1: Blocos de construção CASE [02]

As três camadas inferiores representam o suporte básico para o funcionamento das ferramentas CASE. Devem ser sempre levados em consideração quando da criação de uma nova ferramenta. Os Serviços de Portabilidade provêm suporte aos *frameworks* de integração (Estruturas de Integração). Estes *frameworks* constituem uma coleção de programas com

propósito específico de tornar possível a comunicação entre diferentes ferramentas CASE, criar uma base de dados para o projeto e exibir dados para o usuário final de uma mesma forma. Resumidamente a integração não é vista ou manipulada pelo usuário final. A própria ferramenta, utilizando um suporte de integração, torna isso transparente. É através dos Serviços de Portabilidade e *frameworks* de integração que as ferramentas CASE podem trabalhar de forma integrada, eliminando a dependência de plataforma de hardware e sistema operacional.

A Figura 9.2 mostra as opções de integração entre ferramentas CASE. A ferramenta individual representa na realidade uma solução fechada e particular. Caso esta ferramenta possua integração de dados, como a maioria dos casos, interações com outras ferramentas podem ser levadas em consideração. Nesse nível diferentes ferramentas devem concordar no formato dos dados compartilhados entre elas. Em alguns casos, as ferramentas podem ser projetadas para se integrarem através de uma ponte¹¹. Em termos de uma única fonte, o fabricante integra suas próprias ferramentas num único pacote. Esta integração efetiva comporta-se como uma arquitetura fechada, obstruindo a integração com ferramentas de outros fabricantes. O IPSE (*Integrated Project Support Environment*) é o mais alto nível do espectro de integração. Os padrões para cada blocos de construção CASE são definidos e os fabricantes constroem suas ferramentas compatíveis com estes padrões, sendo portanto compatíveis com ferramentas de outros fabricantes.

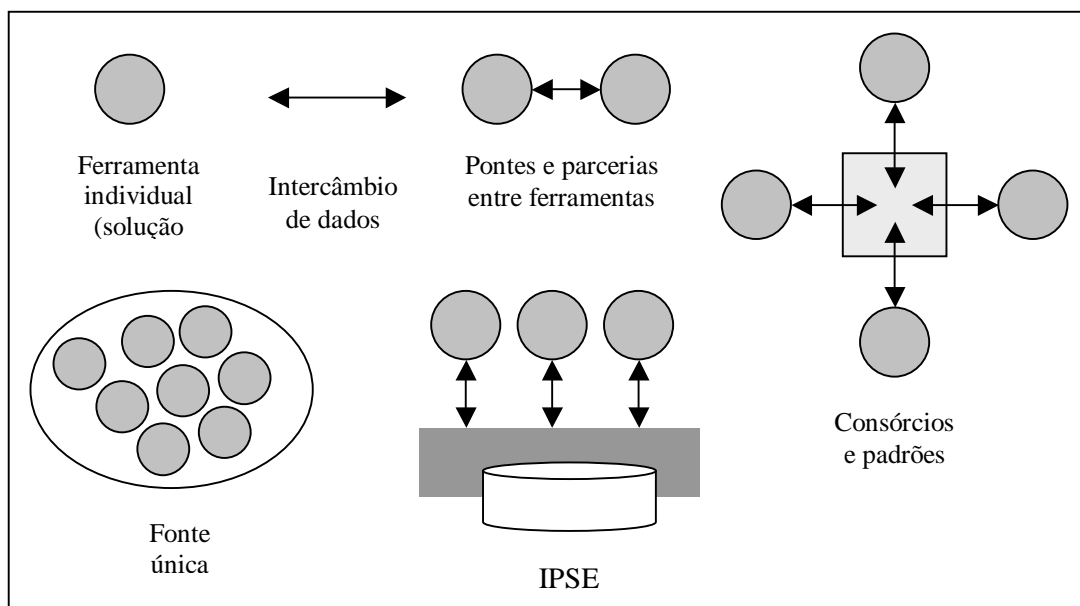


Figura 9.2: Opções de Integração [02]

10. Integração entre ferramentas CASE

Embora a tecnologia CASE tenha importância crucial no desenvolvimento de software, diversos problemas começaram a surgir com seu rápido crescimento. Um dos principais problemas é a integração entre ferramentas de diferentes fabricantes. Muitos são os motivos dessa carência de interoperabilidade, alguns abordados em 7.1.4. Nesta seção vamos falar de forma mais detalhada sobre os níveis e problemas de integração entre diferentes ferramentas CASE, *workbenches* (10.3) e ambientes de integração.

O maior benefício do uso de ferramentas CASE no processo de software pode advir com a integração. Os benefícios de I-CASE (*Integrated CASE*) incluem: (1) eficiência na transferência

¹¹ Uma ponte pode ser algum módulo à parte, encarregado de converter o formato dos dados de uma ferramenta para outra.

de informações entre as ferramentas; (2) redução do esforço em atividades como gerenciamento de configuração, garantia de qualidade, produção de documentos; (3) aumento do controle do projeto e (4) melhoria na cooperação entre membros de um projeto. Naturalmente alguns desafios surgem com I-CASE: demanda de consistência na representação da informação, interfaces padronizadas entre as ferramentas, mecanismos homogêneos para comunicação, abordagem efetiva que torne I-CASE compatível com várias plataformas de hardware e sistemas operacionais.

Existem diversas classificações de integração CASE. Sob o ponto de vista de ferramenta, a integração pode ser [20]:

- *Horizontal* - ferramentas são utilizadas numa mesma fase do processo e podem facilmente trocar seus resultados ou serem influenciadas pelos resultados de outra ferramenta usada na mesma fase.
- *Vertical* – a integração ocorre entre diferentes fases do processo. Tanto para as fases seguintes (*Forward integration*) quanto para os estágios anteriores (*Reverse integration*), procurando manter a consistência entre os documentos.

10.1. Requisitos de integração

Antes de tudo, integração de ferramentas requer primeiro uma integração de métodos [21]. Diferentes métodos aplicados ao longo do ciclo de desenvolvimento têm de concordar. Isto significa que: (1) se diferentes métodos são aplicados para diferentes pontos de vista do problema, resultados devem ser compatíveis no sentido de que eles podem ser combinados para resolver o problema, e (2) os resultados produzidos em uma fase podem ser usados como entrada para métodos da próxima fase.

Para definir integração no contexto de processo de software, é necessário estabelecer um conjunto de requisitos para I-CASE. Conforme visto em [15], um ambiente CASE integrado deve:

- Prover um mecanismo de compartilhamento da informação entre todas as ferramentas contidas no ambiente.
- Propagar a mudança ocorrida em uma informação para outros itens de informação relacionados.
- Prover controle de versão e gerenciamento de configuração para todas as informações da engenharia de software.
- Permitir acesso direto e não sequencial a qualquer ferramenta contida no ambiente.
- Suporte a comunicação entre engenheiros de software.
- Coletar métricas das técnicas e do gerenciamento que possam ser usadas para melhorar o processo e o produto.

10.2. Níveis de integração

Os níveis de integração discutidos neste trabalho foram propostos em [16]. Eles se aplicam a ferramentas contidas em *workbenches* ou em um ambiente de integração.

10.2.1. Integração de plataforma

Significa que as ferramentas e *workbenches* devem ser implementados sobre uma mesma plataforma¹². A maioria das ferramentas CASE estão disponíveis para sistemas UNIX ou PCs rodando Windows. Integração de plataforma em um sistema singular não constitui um grande problema, a menos que exista a necessidade de integrar ferramentas de PCs com UNIX. O

¹² Referindo-se a uma combinação de hardware, sistema operacional ou uma rede de sistemas.

maior problema de integração de plataforma ocorre quando uma empresa possui redes heterogêneas, com diferentes computadores, rodando diferentes sistemas operacionais. Uma simples diferença de versão do sistema operacional pode ser suficiente para o não funcionamento dos sistemas CASE. A melhor solução ainda é adotar sistemas operacionais conhecidos universalmente e largamente utilizados.

10.2.2. Integração de dados

A razão de ser dessa integração é que diferentes ferramentas CASE podem trocar dados. Resultados de uma ferramenta podem ser passados como entrada para outra. Existe um número de diferentes níveis de integração de dados:

1. *Arquivos compartilhados.* Cada ferramenta reconhece um determinado formato de arquivo. O formato de propósito mais geral é um arquivo construído com linhas de caracteres. Qualquer ferramenta pode escrever/ler arquivos nesse formato. Uma pequena otimização desta integração é a possibilidade de fazer os processos se comunicarem através de *pipes*, onde um *stream* de caracteres partindo de um processo para o outro elimina a necessidade de criação de um arquivo intermediário. Esta abordagem simples para troca de informação força os programas a conhecerem a estrutura lógica do arquivo. Funcionando como uma comunicação ponto-a-ponto, esta integração faz com que as ferramentas envolvidas ou concordem no formato dos dados trocados ou possuam filtros que convertam os dados compartilhados numa outra representação. A implementação desses filtros pode representar um aumento de custo significativo, considerando que diferentes ferramentas podem entender formatos bastante distintos, e para cada formato de entrada em uma ferramenta, um filtro deva ser implementado.
2. *Estruturas de dados compartilhadas.* As ferramentas fazem uso de estruturas de dados compartilhadas que normalmente incluem informação de linguagem de design ou programação. Os detalhes sintáticos e semânticos das estruturas são reconhecidos por todas as ferramentas. Estas estruturas podem ser representações de notações como diagramas de fluxo de dados, entidade-relacionamento ou até estruturas de alguma linguagem de programação. Os melhores exemplos desta integração são as ferramentas de conversão entre linguagens de programação. As ferramentas compartilham uma árvore sintática e uma tabela de símbolos que são geradas pelo compilador. Editor, depurador, animador e analisador semântico podem compartilhar as mesmas estruturas. Tais ferramentas geralmente são de um mesmo vendedor, o que torna difícil importar outras ferramentas.
3. *Repositório compartilhado.* As ferramentas são integradas através de um sistema de gerenciamento que inclui um modelo de dados, compartilhado e público, descrevendo as entidades e relacionamentos que podem ser manipulados pelas ferramentas. Este modelo é acessível por todas as ferramentas e conhecido como Sistema de Gerenciamento de Objeto. A integração por repositório compartilhado é a forma mais flexível de integração de dados. Um sistema de banco de dados deve incluir facilidades para tipificar entidades, associar atributos a estas entidades e estabelecer relacionamentos entre elas. Apenas dois pequenos detalhes tornam este modelo de integração desvantajoso: (1) ferramentas devem acessar constantemente o repositório e (2) o usuário deve comprar também o Sistema de Gerenciamento de Objeto, fazendo do repositório um custo adicional. A Figura 10.1 mostra este modelo de integração.

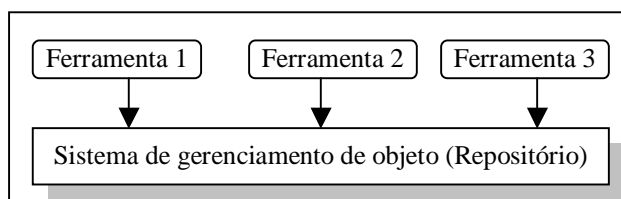


Figura 10.1: Modelo de Repositório Compartilhado

10.2.3. Integração de apresentação

As ferramentas devem usar uma metáfora ou estilo comum e um conjunto de padrões para interação com o usuário. Com isso, os usuários reduzem consideravelmente o *overhead* de aprendizado da ferramenta.

Existem três níveis de integração de apresentação:

1. *Integração de Sistema de Janela.* Ferramentas devem apresentar mesmo sistema de janelas e comandos de manipulação.
2. *Integração de Comando.* As ferramentas devem usar a mesma forma de comandos para funções utilizadas. Tanto em linhas de comando quanto em componentes gráficos, os comandos são os mesmos. Menus são posicionados num mesmo lugar em cada aplicação. Aplicações devem apresentar botões padrões de *Sair/Fechar* etc. Padrões de design encontram-se hoje definidos para que os desenvolvedores sigam uma mesma linha de desenvolvimento baseados na definição de componentes e seus comportamentos.
3. *Integração de Interação.* Aplicada em sistemas com manipulação direta da interface onde o usuário interage diretamente. Por exemplo, se é possível selecionar um texto com um duplo clique, então as demais entidades gráficas também devem ser selecionáveis do mesmo jeito. Se a atividade padrão de selecionar um texto for arrastando o cursor sobre o mesmo, então todas as ferramentas devem fazê-lo da mesma forma. Prover um padrão para este tipo de integração é uma tarefa difícil. As interações são muitas e a quantidade de representações de objetos gráficos e textuais é muito grande.

10.2.4. Integração de controle

Diz respeito a uma ferramenta contida em um *workbench* ou em um ambiente ter algum tipo de controle sobre outras ferramentas no sistema CASE. Ativar, parar ou utilizar os serviços de outras ferramentas são exemplos deste tipo de integração. Estas facilidades podem ser postas em prática através de interfaces públicas de acesso.

Uma outra abordagem para esta integração é a passagem de mensagem entre as ferramentas. Um servidor de mensagens genérico gerencia a comunicação entre as ferramentas e é responsável por repassar a mensagem da ferramenta solicitante para a interface de acesso da ferramenta provedora do serviço. Detalhes de localização das ferramentas e codificação das mensagens são tratados pelo Servidor de Mensagens (Figura 10.2). Ao solicitar um serviço de outra ferramenta, uma mensagem é criada e enviada para o Servidor de Mensagens que por sua vez se encarrega de entregá-la ao destino correto. Isto provê transparência de localidade¹³, podendo o sistema ter características de distribuição de processamento.

Esta integração geralmente é feita por RPC¹⁴ ou através da publicação do formato dos dados e serviços em IDL (*Interface Definition Language*). Os tipos padrões descritos em IDL são conhecidos pelas ferramentas que passam a lidar com uma representação única.

Os novos padrões de comunicação utilizam uma filosofia semelhante. Em CORBA (*Common Object Request Broker Architecture*) [17], por exemplo, os serviços e dados são declarados numa IDL específica e ferramentas auxiliares geram o código necessário para dar suporte à comunicação. No entanto, estas abordagens não resolvem completamente o problema de compartilhamento de estruturas de dados. A natureza distribuída significa memória distribuída, não compartilhada, podendo gerar inconsistências quando da necessidade de se compartilhar uma mesma estrutura entre diferentes ferramentas.

¹³ Solicitantes não têm necessidade de saber onde os provedores de serviço se encontram. Eles apenas fazem uma solicitação para alguém que se encarrega de tratar todos os detalhes de comunicação. Se a ferramenta alvo está ou não numa mesma máquina ou faz parte de outro processo, pouco importa.

¹⁴ *Remote Procedure Call*. Mecanismo de suporte a comunicação entre processos remotos.

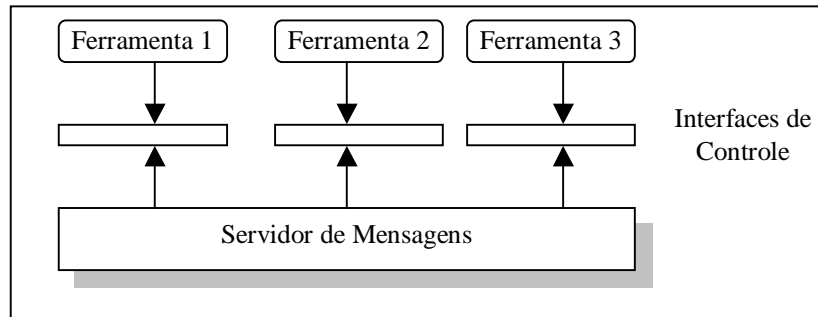


Figura 10.2: Integração por Passagem de Mensagem

10.2.5. Integração de processo

O sistema CASE deve possuir “conhecimento” sobre as atividades do processo, fases, restrições e ferramentas necessárias para suportar estas atividades. Dessa forma, o sistema CASE participa ativamente do agendamento das atividades e da checagem de seu cumprimento. Para isso, o sistema deve conter um modelo de processo de software e utilizá-lo para guiar as atividades executadas. Essa dependência com um modelo de processo pode apresentar alguns problemas indesejados:

- Os modelos de processos são genéricos. As interações entre as atividades não são precisamente definidas.
- Os engenheiros nem sempre seguem à risca o modelo adotado. Às vezes uma pequena mudança no processo de software ocorre durante sua execução. Inserir esse tipo de flexibilidade num modelo é difícil.
- Os modelos especificam os produtos e comunicações entre os desenvolvedores. Já outras tarefas são difíceis de serem especificadas.

Na Figura 10.3, a área sombreada indica a necessidade de interação das ferramentas entre as fases do processo. Ambos os sentidos devem ser levados em consideração na interação entre as ferramentas. Este tipo de integração representa a Integração Vertical (início da seção 10).

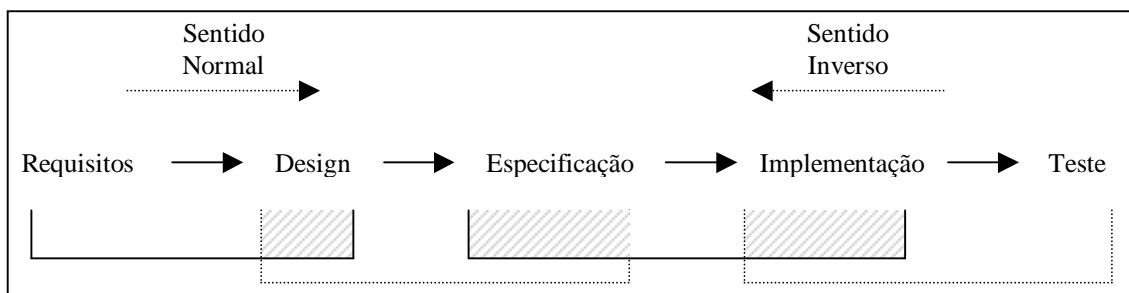


Figura 10.3: Integração no processo de desenvolvimento

10.3. Workbenches

Um *workbench* é um conjunto de ferramentas CASE, reunidas com o objetivo de fornecer suporte a uma fase particular do processo de software. A vantagem de se agrupar ferramentas em um *workbench* é que elas podem trabalhar juntas para prover maior suporte que uma ferramenta singular [03]. Os serviços comuns podem ser implementados e solicitados por outras ferramentas.

As fases do processo de software que mais se beneficiam de *workbenches* são: análise e design, programação e teste. Talvez por serem as fases mais bem compreendidas do processo, isso leve à construção de ferramentas eficientes, que se integram para aumentar a produtividade.

À medida que a integração se torna mais efetiva, os *workbenches* assumem a postura de um sistema aberto. Este comportamento apresenta algumas vantagens:

- Novas ferramentas adequadas para as necessidades particulares de uma organização podem ser acrescentadas ao *workbench*, ou ferramentas antigas podem ser substituídas por alternativas mais adequadas.
- As saídas das ferramentas são arquivos que podem ser manipulados por algum sistema de gerenciamento de configuração.
- Escalabilidade e evolução do *workbench*.
- As necessidades de uma organização são independentes de *workbenches*. A substituição de uma ferramenta inclusa no *workbench* por outra mais adequada implica apenas no desligamento da obsoleta, não comprometendo o uso do conjunto por completo.

10.4. Ambientes de Integração

Workbenches funcionam como ilhas de automação. Cada um deles possui seus próprios dados, dificultando a garantia de consistência no gerenciamento de configuração para todas as saídas do processo de software. Manipular ou utilizar informações de um *workbench* particular em outro pode ser muito difícil e às vezes até impossível [03].

Um Ambiente de Engenharia de Software (*SSE – Software Engineering Environment*) visa resolver as limitações dos *workbenches* e prover maior suporte a todas, ou a maioria das atividades do processo.

A noção de SSE surgiu em 1980, com uma tentativa de projetar um ambiente de suporte à programação em Ada. A partir de então, organizações governamentais e empresas iniciaram um programa de pesquisa para desenvolver ambientes de integração com suporte para todo o processo. Estes ambientes deveriam prover melhoras significantes na produtividade e na qualidade do produto final.

Um SSE pode ser definido da seguinte forma [03]:

“Um conjunto de ferramentas de hardware e software que agem em conjunto, de forma integrada, para prover suporte às atividades do processo de software desde a especificação inicial até os testes e distribuição do produto”

As características básicas que distinguem um SSE de um *workbench* são:

- As facilidades do ambiente são integradas. Os ambientes devem suportar todos os níveis de integração, citados em 10.2.
- Um ambiente é preparado para suportar “times de desenvolvedores” ao invés de um desenvolvedor individual, provendo assim suporte ao gerenciamento de configuração para todas as atividades.
- As facilidades estão disponíveis para suportar um grande número de atividades de desenvolvimento de software. Assim, SSE podem incluir *workbenches* que suportam especificação, design, documentação, programação, teste, depuração etc.

A integração em um SSE geralmente é feita através de um repositório compartilhado com informações de projeto. Todas as ferramentas interagem com o repositório e trocam informações através do mesmo. Isso pode representar uma diminuição significativa dos custos de gerenciamento de configuração, já que todos os documentos do projeto estão armazenados num mesmo lugar e os *links* entre estes documentos podem ser gerenciados mais facilmente.

Para prover o suporte necessário a diferentes projetos, os SSEs devem acomodar *workbenches* e outras ferramentas CASE, possuindo também escalabilidade. Ou seja, sempre que necessário, um SSE deve suportar adição de novas facilidades. Sob esse ponto de vista, um SSE pode ser compreendido como um conjunto de serviços que são usados pelas facilidades de suporte ao usuário final. Estes serviços podem ser oferecidos pela plataforma em que o SSE está

sendo executado ou por um *framework* de serviços. A Figura 10.4 mostra este modelo de ambiente.

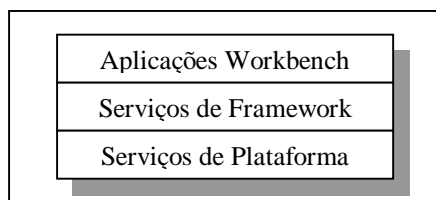


Figura 10.4: Modelo de SSE em níveis

A plataforma em que um ambiente executa é chamada de sistema hospedeiro. Em geral, é constituída por diversos computadores heterogêneos. Cada serviço provido pela plataforma possui facilidades utilizadas pelos ambientes. Sistema de arquivos, serviços de gerenciamento de processos, serviços de rede, serviços de comunicação, serviços de gerenciamento de janelas e serviços de impressão são alguns exemplos de Serviços de Plataforma.

Os Serviços de *frameworks* estendem os serviços providos pela plataforma ao SSE. Normalmente são implementados usando os próprios Serviços de Plataforma e suportam as operações das ferramentas ou *workbenches* presentes no ambiente. A melhor forma de compreender os serviços de *frameworks* é analisá-los em num contexto de um modelo de referência. A Figura 10.5 mostra o modelo IPSE. Este padrão identifica cinco tipos de serviços que um ambiente deve prover:

- *Serviços de Repositório de Dados* – no sentido de prover facilidades para armazenamento de itens de dados e seus relacionamentos.
- *Serviços de Integração de Dados* – visando prover facilidades para gerenciamento de grupos ou configurações dos itens de dados. Juntamente com o Serviço de Repositório formam a base para a integração de dados no ambiente.
- *Serviços de Gerenciamento de Tarefas* – contendo facilidades para a definição e ativação das tarefas presentes no modelo de processo, bem como a verificação de seu andamento. Suportam Integração de Processo (10.2.5).
- *Serviços de Mensagens* – provendo facilidades para comunicação (ferramenta-ferramenta, ambiente-ambiente). Suportam Integração de Controle (10.2.4).
- *Serviços de Interface de Usuário* – possuindo facilidades para desenvolvimento de interfaces de usuário. Suportam Integração de apresentação (10.2.3).

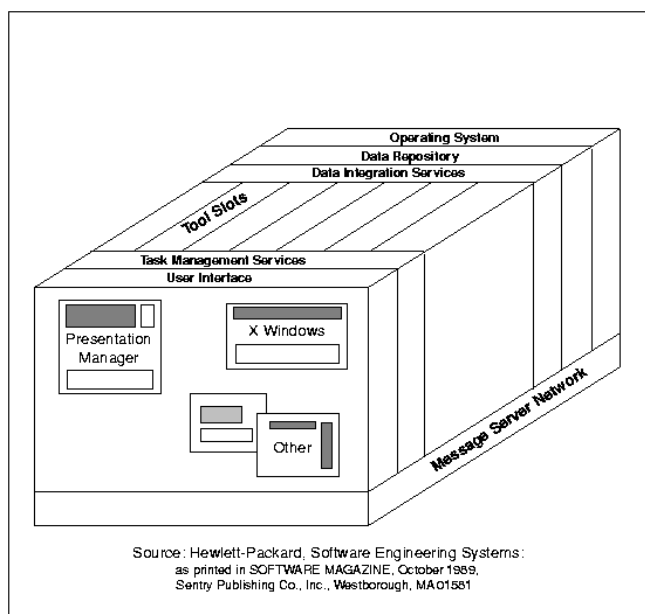


Figura 10.5: Modelo IPSE completo

Obs: O Sistema Operacional encontra-se incorporado ao modelo, porém não deve ser compreendido como parte integrante do mesmo. A presença indica apenas uma relação de dependência entre os Serviços de Framework e os Serviços de Plataforma providos pelo Sistema Operacional.

Finalmente as aplicações *workbenches* ou ferramentas CASE encaixam-se nos *slots* e interagem utilizando todo o suporte provido pelos serviços de *frameworks* e pela plataforma.

11. Integração entre ambientes

Aumentando ainda mais o suporte ao desenvolvimento *in the large*, os ambientes podem ser integrados entre si. Devido à carência de um padrão de integração universal, os problemas são ainda maiores e bem mais complexos do que integrar ferramentas. Um nível a mais na abstração de integração representa um crescimento abrupto na complexidade do problema.

A solução proposta em [24] inclui a criação de um mecanismo de interface entre os ambientes a serem integrados. Problemas comuns com esta abordagem são a incompatibilidade entre linguagens fonte e alvo, múltiplas visões utilizadas nos ambientes, diferentes convenções de nomes e diferentes tipos de representação da informação gráfica.

A estratégia utilizada para construir uma interface entre ambientes consistiu em utilizar o Alchemist, um gerador de transformações entre notações. O usuário descreve as representações fonte e alvo como gramáticas livres de contexto e define um mapeamento entre as regras de cada uma. A especificação do mapeamento é baseada na notação de Gramáticas TT¹⁵. O Alchemist gera o código de transformação (*parser*, *unparser* e *mapper*) a partir da especificação do mapeamento e das notações gramaticais fonte e alvo. A Figura 11.1 mostra o processo de geração.

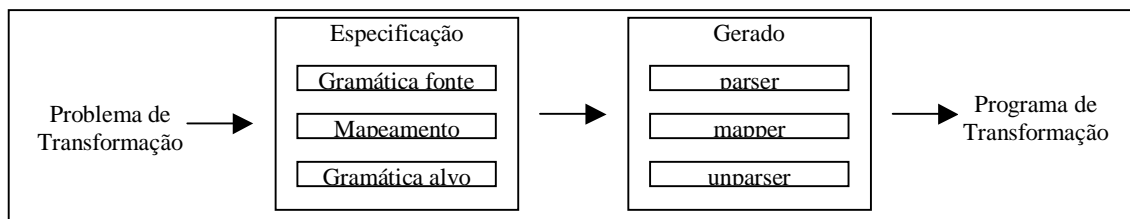


Figura 11.1: Geração da transformação feita pelo Alchemist

Ao final, o programa produzido faz a transformação completa entre estruturas da linguagem fonte para a linguagem alvo (Figura 11.2).

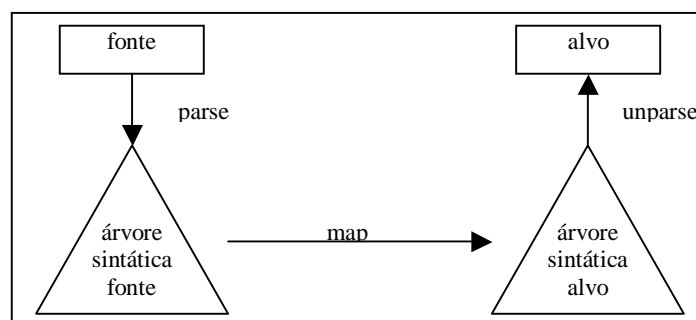


Figura 11.2: Processo de transformação

¹⁵ Um tipo de gramática que gera sub-árvores numa linguagem alvo a partir de sub-árvores numa linguagem fonte.

A idéia apresentada é genérica e foi implementada para as linguagens OCML (*Operationalizable Conceptual Modeling Language*) e Foundation. Assim, o ambiente para design em OCML foi integrado com o Andersen Consulting's Foundation, um outro ambiente CASE comercial.

12. Integração e padronização

Uma das principais razões pelas quais os usuários não adotam ferramentas CASE é a carência de integração/padronização [19]. Alguns padrões emergentes são dinamicamente redefinidos visando tornarem-se *default* no mercado mundial. Diferentes níveis de integração do produto e padronização afetam as decisões dos usuários na adoção das ferramentas CASE. Nesta seção, abordamos alguns problemas e soluções relacionadas com a integração e padronização de ferramentas CASE sob a perspectiva do usuário.

A integração de ferramentas CASE se constitui de diferentes pontos de vista relativos às perspectivas do usuário final. Segundo [19], cinco principais áreas de integração podem ser examinadas cuidadosamente.

12.1. Integração entre ferramentas de um único fabricante

Vista também como uma integração “interna”, este tipo de integração consiste em compartilhar um mesmo dicionário de dados¹⁶ entre as ferramentas. Pode ser considerada como uma integração de natureza proprietária.

12.1.1. Problemas

Esta abordagem limita os usuários às ferramentas fornecidas por um fabricante específico. A maioria deles não oferece ferramentas para o ciclo de vida completo, nem produz ferramentas para diferentes plataformas (PC, *workstations* e *mainframes*). Dependendo do usuário final, isto pode representar um item primordial no processo de adoção/escolha.

Algumas ferramentas, embora de um mesmo fabricante, utilizam um dicionário de dados local para armazenar as informações compartilhadas. Problemas de sincronização de acesso e incompatibilidade podem surgir com esta abordagem.

Utilizar bases de dados relacionais para representar dicionário de dados pode não acomodar corretamente objetos e tipos flexíveis e complexos (ex. segmentos de código, diagramas de design, processos de usuários) requeridos pela tecnologia CASE.

12.1.2. Soluções

Alguns fabricantes estão expandindo o conceito de dicionário de dados. Isto ajudará a minimizar o problema de conciliação da base de dados, embora possa afetar negativamente a performance das ferramentas, caso não seja implementado corretamente. Os esforços atuais estão caminhando no sentido expandir a integração para bases de dados orientadas a objetos (OODB). Diferentemente das relacionais, estas bases armazenam dados e regras de acesso juntamente com o objeto.

¹⁶ Geralmente uma base de dados relacional que provê armazenamento dos dados e acesso para as ferramentas.

12.2. Integração entre ferramentas de diferentes fabricantes

Ocorre quando um vendedor integra ferramentas com as de um outro vendedor. Conhecida também como integração “externa”, ela pode ser tornar possível através de controle de acesso e/ou controle de dados. No primeiro deles, o vendedor insere facilidades nas ferramentas para serem acessadas/controladas por outras. No controle de dados, as ferramentas externas manipulam diretamente os dados contidos no dicionário interno de outras ferramentas.

12.2.1. Problemas

Muitos fabricantes se importam em publicar interfaces de acesso aos dados e serviços de suas ferramentas com o intuito de promovê-la como uma ferramenta útil e integrável. Entretanto nem sempre o formato dos dados é completamente descrito e a incompatibilidade entre as ferramentas persiste. Facilidades de exportação/importação preocupam-se apenas com os problemas dos dicionários de dados que não podem ser compartilhados. Os dados podem não ser exportados da melhor forma e uma importação pode não capturar corretamente a verdadeira expressividade dos dados. Problemas típicos dessas atividades são formatos, tipos e convenções de nomes.

Um outro problema reflete diretamente a importância da seleção e aceitação de padrões. Compatibilizar ferramentas de diferentes fabricantes requer um custo adicional em desenvolvimento, suporte e manutenção. Os custos crescem exponencialmente com o número de interfaces, plataformas e ambientes suportados pelos outros fabricantes [19].

12.2.2. Soluções

Muitos fabricantes estão se preparando para suportar o conceito de repositório, embora mantenham suas bases proprietárias como um produto à parte. A abordagem do repositório, mesmo sendo um padrão de fato, permite usuários utilizarem efetivamente diferentes ferramentas até que se escolha a mais apropriada para suas necessidades. Parcerias estratégicas têm sido formadas para tentar estabelecer um padrão de ferramentas CASE, no entanto nenhuma delas obteve impacto significativo nos processos de desenvolvimento comumente adotados.

12.3. Integração de ambiente operacional

Algumas ferramentas possuem versões que rodam em PC, enquanto outros são direcionados a *workstations* e *mainframes*. A escolha do sistema geralmente está relacionada com a natureza técnica (Sistemas de Tempo Real) ou comercial (Sistemas de Gerenciamento de Informação) da aplicação.

12.3.1. Problemas

As ferramentas podem usar feições específicas do sistema operacional ou utilizar recursos indisponíveis em outros sistemas operacionais (ex. sistemas de janelas específicos, variantes do sistema UNIX). Isto dificulta ou impossibilita a utilização dessas ferramentas em outro ambiente operacional.

Problemas de escalabilidade podem surgir quando da migração da ferramenta entre plataformas. Algumas ferramentas podem sofrer degradação de performance devido às limitações no novo ambiente (ex. memória, CPU, facilidades de armazenamento). Em PCs pode ser impossível executar múltiplas instancias da ferramenta e o gerenciamento de pedidos de acesso à base de dados pode não ser bem gerenciado.

Um outro problema é com os sistemas de gerenciamento de configuração. Algumas ferramentas integram um simples controle de versão baseado em arquivos, excluindo os tipos de dados e objetos complexos.

12.3.2. Soluções

Uma possível solução é o padrão do UNIX, mostrando o compromisso de um sistema operacional suportando as necessidades tanto do mercado comercial quanto técnico [19]. As funções de suporte a rede (ex. NFS, TCP/IP) podem representar um diferencial significativo para a integração de ferramentas CASE.

A Sun Microsystems tem revelado seu interesse no mercado CASE com a introdução de NSE (*Network Software Environment*), um sistema de gerenciamento de configuração proprietário, convidando inclusive demais fabricantes a desenvolverem ferramentas para sua plataforma.

12.4. Integração de processo de desenvolvimento

Ferramentas CASE são adotadas em todas as fases do processo de software. Elas não são mais dirigidas especificamente fases de análise e design. O objetivo é combinar várias fases do ciclo de vida do software e gerenciar a transação entre as mesmas. Alguns fabricantes estão analisando facilidades do IPSE e I-CASE. O IPSE oferece integração de dados, controle e apresentação, enquanto I-CASE estão centrados em uma enciclopédia compartilhada. Em termos de complexidade, a enciclopédia pode ser um dicionário ou um repositório. Significados associados com objetos são armazenados e usados para derivar outros objetos, embora a enciclopédia não seja necessariamente projetada para uma tecnologia orientada a objetos.

12.4.1. Problemas

O primeiro problema é a flexibilidade. Usuários têm requerido que ferramentas apresentem facilidade de modificação e adaptação a seus processos e métodos particulares. A maioria das ferramentas não permite modificação. As conhecidas como “abertas” permitem geralmente modificações nas características de apresentação. Para permitir integração completa, os usuários devem ser capazes de modificar o comportamento da ferramenta (ex. regras de design, tipos de objetos, processos) e não apenas interface de usuário [15].

Um outro problema é que a maioria das ferramentas CASE suportam metodologias de design específicas, geralmente inadequadas para uma organização. As ferramentas deveriam apresentar flexibilidade de expansão das abordagens de design e permitir modificações de configuração para suportar métodos alternativos.

12.4.2. Soluções

Diversas opções de ferramentas e metodologias tornaram-se disponíveis. AD/Cycle da IBM é um exemplo de combinação de métodos, tecnologia de armazenamento central e ferramentas de suporte CASE. Conjuntos de métodos e consistência de repositório são oferecidos pelo fabricante.

Trabalhos com IPSE e I-CASE sofreram grande progresso. Neste último, os esforços foram enfocados no mercado comercial de Sistemas de Gerenciamento de Informação. A ênfase principal foi a integração de fases *front-end* do ciclo de vida que dão suporte à geração de código (ex. planejamento, análise, design). A Hewlett-Packard tem se envolvido na definição de um *framework*. O produto não provê apenas fundamentação para integração entre ferramentas CASE da HP, mas também com outras ferramentas, através de um protocolo de interface.

12.5. Integração de usuário final

A maior ênfase na integração de ferramentas CASE diz respeito aos próprios usuários. O conceito de integração varia de algo tão simples quanto manter a consistência da interface de usuário, até tarefas complexas como prover suporte para adição de novas funcionalidades no design.

12.5.1. Problemas

O aprendizado da ferramenta requer um investimento significativo no processo de adoção e geralmente é maior do que o custo da ferramenta. Treinamento e suporte trazem custo adicional e consomem tempo dos usuários.

12.5.2. Soluções

Os fabricantes têm se preocupado em desenvolver ferramentas com interfaces de usuário padronizadas. Isso pode reduzir sensivelmente o tempo de aprendizado. Padrões como X-windows têm sido bem aceito no mercado e mostram como uma interface de usuário particular pode se tornar importante no processo de padronização.

Existe um consenso geral de que a interface de usuário contribui significativamente para a aceitação e tempo de aprendizado de um produto. Fabricantes devem ter cuidado para não promover uma solução rápida e de baixa qualidade. Se eles não analisarem cuidadosamente os requisitos de interface de usuários, sem levar em conta o contexto do uso da ferramenta, podem criar um padrão de interface que promove consistência e prejudica a facilidade de uso.

12.6. Fontes de padronização

Conforme foi previamente mencionado, um dos maiores obstáculos para a aceitação de ferramentas CASE por parte dos usuários é a falta de padronização. Usuários não estão dispostos a gastar tempo e esforço para adotar uma ferramenta e depois descobrir a incompatibilidade com o processo adotado e com os padrões consolidados.

Em 1986 um estudo mostrou que existiam cerca de 250 trabalhos de padronização em progresso, dos quais 19 deles eram específicos para ferramentas CASE [19].

A atitude dos fabricantes denota uma certa contraposição à padronização. Eles não estão dispostos a suportar padrões específicos ou participar de esforços conflitantes para definir os melhores padrões. Ao invés disso, preferem apostar que suas interfaces de integração tornam seu produto independente de padrão e sempre vai ter algum grau de compatibilidade com o padrão mais aceito.

Todo o esforço de padronização é confundido pelo fato de que diferentes padrões originariam diferentes interfaces. Alguns deles são focados na especificação de repositórios, outros em portabilidade, outros em troca de dados. Diferentes pontos de vista estão conduzindo os padrões a papéis similares que provavelmente resultarão em definições incompatíveis [19].

Algumas questões de padronização são esquecidas pelos diversos comitês. A primeira é a questão das métricas. Um conjunto de métricas bem definidas poderia ser desenvolvido e utilizado para quantificar os resultados. Isto ajudaria os usuários a medir seus progressos na produtividade e qualidade. Uma outra questão é a padronização de relatórios. Qualquer formato de relatório deve definir mais do que como imprimir diagramas de design. Ele deve criar um *framework* para extração de informações sobre o ciclo de desenvolvimento (ex. métricas de uso, histórico de gerenciamento de configuração, relacionamentos entre objetos).

13. Esforços de padronização

Nesta seção, citamos rapidamente as principais tentativas de padronização de ferramentas CASE, com o intuito de fornecer uma breve visão dos mais conhecidos e respeitados trabalhos.

ATIS/CATIS/CIS

O padrão ATIS (*A Tool Integration Standard*) foi originalmente desenvolvido pela Atherton Technology para definir a interface comum do repositório de dados da companhia Software BlackPlane. Atualmente ATIS define a interface orientada a objetos para o repositório CDD/Plus da empresa DEC.

O CATIS (*Common Application and Tools Interface Standard*) é um padrão baseado no ATIS, mas focalizado nas propriedades de um sistema de gerenciamento de objetos (ex. distribuição de objetos, tipos de dados flexíveis, controle de versão).

O CIS (*CASE Integration Services*) é uma variação do CATIS que suporta a abordagem orientada a objetos, mas se diferencia nos mecanismos de controle (controle distribuído, ao invés de interação com um único repositório).

CDIF

O padrão CDIF (*CASE Data Interchange Format*) é uma extensão de um outro padrão, EDIF (*Electronic Data Exchange Format*), usado para troca de dados entre ferramentas CAD. Este padrão é focado na especificação de requisitos para integração por troca de dados entre ferramentas CASE e foi originalmente proposto pela Cadre Technologies, Inc.

IEEE(P1175)

É um padrão de interconexão entre ferramentas proposto pela IEEE. A transferência de informação entre as ferramentas é tratada com detalhes neste padrão. Uma linguagem para troca de informações, a STL (*Standard Text Language*), descreve a informação que trafega entre as ferramentas.

IRDS(ANSI/ISO)

O IRDS (*Information Resource Dictionary Standard*) concentra-se na fonte de integração de dados via repositório centralizado. Este padrão define a interface do dicionário de dados para uma base relacional. Duas versões deste padrão foram desenvolvidas: uma aprovada pela ANSI (1988), baseada no modelo entidade-relacionamento e outra promovida pela ISO, suportando diferentes modelos de dados.

PCTE/PCTE+

O padrão PCTE (*Portable Common Tools Environment*) é um padrão europeu dirigido para definição de interfaces de ferramentas de suporte para um IPSE. Foi originalmente projetado para suprir as questões de interoperabilidade de ferramentas e dados entre ambientes. Ele define interfaces que provêm bases de dados distribuídas, arquitetura distribuída e interface de usuário estendida e uniforme. Sua primeira versão foi publicada em 1984 e inicialmente não despertou o interesse comercial.

Devido ao número de deficiências técnicas do PCTE, foi desenvolvido o PCTE+, adicionando-se ainda segurança, controle de versão e sistema de gerenciamento de objetos.

SIGMA

O padrão SIGMA (*Software Industrialized Generator and Maintenance Aids*) é focado em todos os níveis de integração de ferramentas (ex. portabilidade, troca de dados, arquitetura de repositório). Foi projetado para criar ambientes de desenvolvimento de softwares independentes de adaptações. O padrão foi desenvolvido em parceria pelo governo e indústria Japoneses. Um protótipo foi concluído em 1989 como parte do planejamento de um projeto de 5 anos e posto em comercialização.

IPSE

O IPSE (*Integrated Project Support Environment*) foi uma abordagem proposta no sentido de prover melhor suporte para interoperabilidade entre ferramentas CASE [22]. Inicialmente as ferramentas deveriam ser desenvolvidas para interagir através do uso de serviços do sistema operacional (ex. sistema de arquivos, chamadas de procedimentos remotos) ou através de uma base de dados comum. O padrão se concentrou mais na integração de dados, através de um repositório central mantendo informações, mecanismos de consulta, granularidade, localização física, controle de versão, *triggers* etc. Demais facilidades poderiam ser desenvolvidas como uma camada acima do repositório.

14. Conclusão

A produção de software tem tomado dimensões de produção em larga escala, no sentido de que metodologias e ferramentas auxiliares são usadas para aumentar a produtividade e melhorar a qualidade do produto, mostrando assim que o desenvolvimento de software é mais do que uma “arte” e merece ter sua devida importância.

A Crise de Software passou a ser vista como um problema global, carente de uma reestruturação nos modelos de desenvolvimento, maior disciplina dos desenvolvedores, tratamento sistemático, abordagem formal e ferramentas de apoio.

Com o aumento de produtividade pelo uso de ferramentas CASE, o mercado de software começou a demandar cada vez mais ferramentas de apoio ao processo de produção. Isso acarretou no surgimento de diversas opções limitadas e com problemas de integração.

Os sistemas CASE possuem um enorme potencial se usados corretamente [14], no entanto, diante de um cenário com diversas opções, a integração entre as ferramentas CASE tornou-se um problema digno de ser discutido com seriedade. Como o mercado é muito comprometedor, cada fabricante procura estabelecer seu padrão de integração. Até mesmo as opiniões de estudiosos do assunto são divergentes. Em [22], acredita-se que a integração se dará com diversos ambientes CASE federados e não com ferramentas e *frameworks* particulares. Já em [24], num relato de experiência prática, acredita-se que um padrão geral de integração deva demorar um bom tempo para surgir e que, por enquanto, problemas de integração de ambientes são resolvidos de forma *ad hoc*. Em [23], aposta-se que, através do uso de arquiteturas de ambientes “abertos”, pode-se diminuir o egocentrismo das novas ferramentas. Em [19], aposta-se nas grandes parcerias entre os fabricantes para encontrar um padrão capaz de satisfazer as necessidades do mercado.

De qualquer forma, a necessidade de padronização tem feito as empresas chegarem a alguma convergência de opinião. Enquanto não se define um “denominador comum”, o mercado dita as regras e os usuários limitam-se apenas a avaliar muitos itens (seção 7) antes de adotar uma ferramenta CASE.

Esperamos que este trabalho sirva como um documento introdutório aos curiosos em conhecer, adotar, avaliar ou desenvolver ferramentas CASE. O futuro aponta para a integração como chave principal para a maximização do uso da tecnologia CASE. Mostramos os eventuais tipos, níveis e problemas de integração como forma a conscientizar interessados sobre os desafios envolvendo Engenharia de Software apoiada por ferramentas.

Referências

- [01] Tom Flecher & Jim Hunt. *Software Engineering and CASE. Bridging the culture gap*, McGraw- Hill, Inc. 1993.
- [02] Roger S. Pressman. *Software Engineering. A practitioner's approach*, McGraw-Hill, Inc. Fourth Edition, 1997.
- [03] Ian Sommerville. *Software Engineering*, Addison-Wesley. Fifth Edition, 1996.
- [04] Itana Maria de Souza Gimenes. *Ferramentas CASE*, Departamento de Informática, Universidade Estadual de Maringá, 1995.
- [05] Aubrey Moore. *Definitions of CASE, CASE Tools and CASE Environments*, Sheffield Hallan University, 1996.
Disponível na Internet:
<http://www.shu.ac.uk/schools/cms/student/amoore/case/case1.html>.
- [06] Robin Milner. *Is Computing an Experimental Science?* Technical Report ECS-LFCS-86-1, University of Edinburg, Departament of Computer Science, Edimburg, UK, 1986, pp. 1 – 7.
- [07] R.W. Floyd. *Assigning Meaning to Programs*. Proceedings Symposium on Applied Mathematics, 19: 19 – 32, 1967.
- [08] C.A.R. Hoare. *An Axiomatic Basis for Computer Programming*, Communications of the ACM, 12: 576 – 580, 1969.
- [09] Nico Plat. *Experiments with Formal Methods in Software Engineering*, Proefschrift Technische Universiteit Delft, Netherlands, 1993, pp. 17–32.
- [10] Aubrey Moore. *CASE: Computer Aided Software Engineering . Brief Outline of CASE History*, Sheffield Hallan University, 1996.
Disponível na Internet:
<http://www.shu.ac.uk/schools/cms/student/amoore/case/caseh.html>.
- [11] Software Engineering Institute. *How do you define Software Architecture?* Carnegie Mellon University, 2001.
Disponível na Internet em: <http://www.sei.cmu.edu/architecture/definitions.html>
- [12] C. Huff & D. Smith & E. Morris & P. Zarrella. *How do you define Software Architecture?* Technical Report CMU/SEI-92-TR-6, Proceedings of the CASE Management Workshop, Software Engineering Institute, Carnegie Mellon University, 1992.
- [13] K. S. Oakes & D. Smith & E. Morris. *Guide to CASE Adoption*. Technical Report CMU/SEI-92-TR-15, Software Engineering Institute, Carnegie Mellon University, 1992.
- [14] W. S. Humphrey. *CASE Planning and the Software Process*. Technical Report CMU/SEI-89-TR-25, Software Engineering Institute, Carnegie Mellon University, 1989.
- [15] G. Forte. *In Search of the Integrated Environment*. CASE Outlook, March/April 1989, pp. 5 – 12.

- [16] A. I. Wasserman. *Tool Integration in Software Engineering Environments*. In Proc. Int. Workshop on Environments, Berlin, 137 – 149 [512], 1990.
- [17] R. Orfali & D. Harkey. *Client/Server Programming with Java and CORBA*. Wiley Computer Publishing. Second Edition, 1998.
- [18] Sandro R. B. O. *Integrando ferramentas em ambientes centrados no processo*. Trabalho Individual em Engenharia de Software, Centro de Informática, UFPE, 2000.
- [19] P. F. Zarrella. *CASE Tool Integration and Standardization*. Technical Report CMU/SEI-90-TR-14, Software Engineering Institute, Carnegie Mellon University, 1990.
- [20] K. Kurbel & T. Schnieder. *Integration Issues of Information Engineering Based I-CASE Tools*. Working Paper n° 33. University of Münster, Institute of Business Informatics, Germany, 1994.
- [21] D. Robinson. *Case Support For Large Systems*. Proceedings of the 3rd European Software Engineering Conference, Lecture Notes in Computer Science 550. Berlin, Heidelberg et al. 1991, pp 504 – 508.
- [22] A.W. Brown & P.H. Feiler & K.C. Wallnau. *Understanding Integrations in a Software Development Environment*. Technical Report CMU/SEI-91-TR-31, Software Engineering Institute, Carnegie Mellon University, 1992.
- [23] K.C. Wallnau & P.H. Feiler. *Tool Integration and Environment Architectures*. Technical Report CMU/SEI-91-TR-11, Software Engineering Institute, Carnegie Mellon University, 1991.
- [24] A.I. Verkamo & G. Lindén. *Problems in Interfacing Tools of Different Development Environments*. In Proceedings of the Seventh International Conference on Software Engineering and Knowledge Engineering, June 1995, pp. 429 – 437.

Apêndice

Este Apêndice contém o significado das principais siglas utilizadas neste documento.

ATIS	A Tool Integration Standard
BNF	Backus Naur Form
CASE	Computer Aided Software Engineering
CATIS	Common Application and Tools Interface Standard
CDIF	CASE Data Interchange Format
CIS	CASE Integration Services
CORBA	Common Object Request Broker Architecture
DoD	Department of Defense
I-CASE	Integrated CASE
IDL	Interface Definition Language
IPSE	Integrated Project Support Environment
IRDS	Information Resource Dictionary Standard
MIS	Management Information Systems
NFS	Network File System
NSE	Network Software Environment
OCML	Operationalizable Conceptual Modeling Language
OODB	Object-Oriented Database
PCTE	Portable Common Tools Environment
PEI	Production Efficiency Index
QSM	Quantitative Software Management
RPC	Remote Procedure Call
RUDE	Run, Understand, Debug, Edit
SEI	Software Engineering Institute
SIGMA	Software Industrialized Generator and Maintenance Aids
SPEI	Software Production Efficiency Index
SSE	Software Engineering Environment
STL	Standard Text Language