

DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO DE SUPORTE INFORMÁTICO

*J. M. Vilas-Boas da Silva **

1. INTRODUÇÃO

Há já alguns anos tudo o que se tinha que fazer para trabalhar com um computador era programá-lo. De facto, nos anos 60, as aplicações informáticas eram concebidas sem o auxílio de uma metodologia explícita de desenvolvimento de sistemas de informação. Nessa altura, a ênfase na programação e na preparação e treino técnicos eram bastante apreciados, em detrimento da capacidade de comunicação e de entendimento das necessidades dos utilizadores. A questão, no fim dos anos 70, princípios da década de 80, ainda oscilava entre aqueles que pensavam que o desenvolvimento de um sistema se confinava ao par de horas que o filho adolescente demorava a criar pequenas aplicações com o seu *Spectrum* e entre os que argumentavam que o desenvolvimento de um sistema comercial, pela sua dimensão e complexidade, não era simplesmente uma aplicação maior, mas requeria *um método de abordagem*.

Ashworth (1990) ilustra a necessidade do método com o exemplo de construção de uma casa. Aqui várias possibilidades se podem colocar ao potencial aspirante ao domicílio:

- i) Poderá, por exemplo, constituir-se como candidato a construtor, comprar os materiais e iniciar a obra por onde a sua sensibilidade ditar. O resultado, longe de espectacular, pautar-se-á seguramente por deficiências de ordem variada, desde a possível ausência de estrutura, passando pelas paredes demasiado instáveis e fracas para suportarem o telhado, até ao provável esquecimento da introdução dos canos da água nas paredes! A causa principal destas falhas será a ausência de planeamento.
- ii) Outra possibilidade era o recurso a alguém com experiência em construção civil, a quem se definiam as exigências para a casa, como sendo: 4 assoalhadas, 1 cozinha, 2 WC e 1 sala. Provavelmente, o resultado também não

* Instituto Universitário de Desenvolvimento e Promoção Social — Pólo de Viseu da Universidade Católica Portuguesa.

iria estar de acordo com as nossas expectativas, pois, ao assumirmos a condição de *expert* do construtor, esquecemo-nos de especificar os nossos reais requisitos, de forma completa e clara.

- iii) Finalmente, a última alternativa passava pela contratação de um arquitecto que ajudasse a definir as verdadeiras necessidades do cliente, por aproximações sucessivas. Neste processo iterativo o cliente iria progressivamente especificando as suas pretensões, de acordo com as restrições legais e físicas existentes. O resultado seriam diversos cenários, de entre os quais acabaria por ser escolhida a solução.

Um *software*, pelo facto de ser um produto com características de maior intangibilidade do que uma casa, não deixa de ter que ser previamente planeado, isto é, de ter que obedecer a um projecto. Por outro lado, garantir a qualidade desta especificação significa assegurar a conformidade entre as reais necessidades do utilizador e as características do produto especificado no projecto. Isto significa tão-somente que o cliente não se pode demitir das suas responsabilidades na definição do produto, sob pena de vir a ser mal servido. Acresce ainda que, os custos desta potencial ameaça, num sistema de gestão integrada da informação à escala da empresa, não são meramente correspondentes aos custos do produto. Estes poderão ser ultrapassados várias vezes pelos prejuízos causados ao bom funcionamento da organização, pelas perdas de produtividade e de competitividade da empresa, aspectos que, em situações mais drásticas, poderão mesmo colocar em causa a sua sobrevivência.

Posto isto, não será de estranhar que o tal método anteriormente referido promova uma abordagem modular, identificando outros estágios no ciclo de vida de desenvolvimento do sistema de informação, para além da *análise das necessidades* do utilizador, fase que, reconhecidamente, foi muito ignorada e evitada, pois assim os especialistas tinham toda a liberdade de acção e os utilizadores recolhiam-se à cómoda posição de dócil plateia ignorante.

Estes estágios vão desde a *análise dos requisitos* até à *manutenção do sistema*, passando pela *especificação*, *projecto do sistema*, *projecto detalhado* e *implementação*. Todos eles deverão possuir metas claras, bem definidas e objectivas que permitirão a avaliação e o acompanhamento por parte do cliente, quer do estado de avanço do projecto, quer da conformidade entre os *outputs* reais de cada fase e os planeados, o que resulta numa eficaz gestão do projecto e numa eficiente utilização dos recursos afectados.

O recurso a métodos estruturados garante também a identificação de actividades perfeitamente definidas, bem como a sua interacção. Ao utilizarem técnicas diagramáticas e de modelação, estes métodos promovem a melhoria efectiva da comunicação entre utilizadores e projectistas, permitindo ainda análises cruzadas, o que garante a consistência e a convergência de várias perspectivas, a adequação

e conformidade do sistema e, finalmente, minimiza os custos de manutenção. Este último aspecto resulta de que as intervenções correctivas são realizadas mais cedo e de que existe todo um conjunto de documentos descritivos e gráficos que suporta e documenta o processo de desenvolvimento. Finalmente, os métodos estruturados promovem uma abordagem formal, sistemática e objectiva com recurso a diversas técnicas devidamente integradas e articuladas num contexto de desenvolvimento de sistemas, o que disciplina e normaliza a actuação, orienta a acção, evita dependências pessoais, confere transparência e responsabilidade ao processo, possibilita o seguimento, a avaliação e a efectiva gestão do projecto, criando ainda formas de medida do sucesso.

Uma metodologia, contudo, para além de uma colecção de procedimentos, técnicas, ferramentas e documentação de suporte à implementação do sistema de informação, comporta uma perspectiva filosófica e persegue determinados objectivos. Exemplos de filosofias são aquelas que advogam como boa solução a utilização maciça de computadores, as que optam pela produção de documentação em grandes quantidades, as que defendem a minimização dos custos, as que defendem a rápida implementação dos sistemas, as que preferem as soluções mais adaptadas aos problemas, as que enfatizam a utilização das técnicas e ferramentas disponíveis, ou as que destacam a importância das afinidades entre o sistema e os utilizadores.

Quanto aos objectivos perseguidos por diversas metodologias, Avison (1994) cita, entre outros, os seguintes:

- *levantamento e registo exactos das necessidades* dos utilizadores, de modo perceptível, por analistas e utilizadores;
- *facultar um método de desenvolvimento sistemático* que permita a fixação de metas temporais e de custos, permitindo a monitoria, a avaliação do progresso e o controle eficaz;
- *produção de um sistema bem documentado e fácil de manter*, face às mudanças organizacionais;
- *introdução de modificações, tão cedo quanto possível*, no processo de desenvolvimento, proporcionando assim a minimização dos custos globais do sistema;
- *proporcionar um sistema amigável* aos agentes que com ele interajam, sejam eles utilizadores, clientes, gestores, auditores ou especialistas responsáveis pela sua manutenção.

Pode pois concluir-se que, à medida que os sistemas de informação informatizados se tornaram progressivamente populares, no meio dos gestores, três principais mudanças passaram a ser fortemente requeridas. A primeira consistiu numa crescente valorização das fases respeitantes à análise e ao projecto dos sistemas e, consequentemente, ao aumento da importância da função do analista, em detri-

mento da do programador. A segunda pautou-se pelo acentuar da necessidade de integração e do desenvolvimento integrado dos sistemas, em detrimento das soluções pontuais, à medida que as organizações cresciam em dimensão e complexidade. Finalmente, e também como consequência dos aspectos acabados de referir, cresceu e consolidou-se o interesse por metodologias de desenvolvimento de sistemas de informação.

A partir do que se afirmou, não é difícil inferir a importância desta temática para os gestores *profissionais* das instituições que, de certeza, por várias vezes na sua vida profissional, se hão-de ver envolvidos em processos de reformulação dos sistemas de informação de apoio à gestão, quer na vertente de participação em *task forces*, quer pela colaboração a dispensar a pedido dos especialistas, quer pela própria actividade de gestão destes projectos, quer ainda pela natureza da sua função dentro da linha orgânica, em que necessitam de saber fazer-se ajudar pelo recurso a especialistas. Para tal, deverão possuir o conhecimento mínimo que lhes permita identificar os problemas, formulá-los e dispor de capacidade de comunicação e entendimento com os especialistas. Portanto, a sensibilização dos gestores para a temática será uma das finalidades deste artigo. Outra, será a ponderação crítica dos procedimentos e metodologias expostos, à luz das suas maiores fragilidades e dificuldades, de modo a que os seus utilizadores também estejam conscientes dos seus pontos fracos que, necessariamente, deverão ser combatidos e ultrapassados.

2. METODOLOGIAS DE DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO

2.1. Principais problemas

Os computadores são máquinas recentes quando comparados com outras. No entanto, nos seus cerca de 50 anos de vida, apresentam taxas de evolução e desenvolvimento muito elevadas, o que faz com que alguns dos seus problemas sobressaíam com uma ênfase particularmente dramática.

Acontece frequentes vezes que projectos informáticos resultam em produtos fracos, incorrem em sobrecusto, geram atrasos significativos e são exemplos de deficiente automatização de procedimentos e de mau funcionamento. Quantos de nós não recebemos já ordens de pagamento no valor de 0\$00 e múltiplas e repetidas cópias de documentos? Quantos de nós não fomos já impedidos de tratar de um assunto, ou fomos erradamente tributados nos impostos, por erros informáticos? Se bem que, algumas vezes, estes problemas se devam a falhas no *hardware*, na maior parte das ocasiões isso não acontece. Como tal, é necessário que se evitem situações e sentimentos de embaraço, alívio, insegurança, incerteza, opti-

mesmo ou pessimismo, associadas ao funcionamento destas máquinas, através do desenvolvimento de metodologias apropriadas que garantam disciplina na abordagem e uma progressão faseada, sequencial e segura quando se avança no processo de desenvolvimento dos sistemas de informação. Estas metodologias dão por diversos nomes, a saber: *Software Engineering*, *Structured Systems Analysis and Design Methodology*, *Information Engineering*, *Business Systems Planning*, *Structured Systems Analysis*, *Jackson Systems Development*, *Information Systems Work and Analysis of Changes*, etc.

Brooks (1982) salienta as dificuldades originadas pelo facto do meio em que se desenvolve a programação ser aparentemente fácil de tratar, apresentando, à partida, poucas limitações. No entanto, facilmente se verifica uma certa frustração, pois as ideias e o pensamento humanos contêm falhas e mesmo erros, o que gera problemas no desenvolvimento do *software*. Por exemplo, qualquer sequência lógica, mesmo que hilariante, ou «pouco lógica» pode ser especificada e, mais ainda, aceite pelo computador que a executará fielmente, sem qualquer objecção. Em resumo, pode afirmar-se que o desenvolvimento de *software* se desenrola num ambiente, no mínimo, estranho, em que nada se pode ter como garantido e em que a probabilidade de ocorrência de erros é elevada.

Thomas (1994) salienta outro tipo de problemas, nomeadamente a complexidade, a dimensão, os prazos, o custo e a gestão dos projectos.

A complexidade é mais fácil de identificar do que de descrever. Embora possua uma vertente relacionada com a dimensão, é mais do que isso: uma tarefa é considerada mais complexa do que outra se a sua descrição é mais expressiva. Como exemplo, a tarefa «somar dois números e imprimir o resultado» é menos complexa do que a tarefa «aceitar dois números, verificar que se encontram dentro de um certo intervalo, somá-los e decidir se este resultado é válido face a um certo critério; não o sendo, o resultado será uma mensagem de erro com um determinado significado; caso contrário, o resultado deverá ser impresso numa listagem formatada e enviado para o terminal vídeo». A complexidade aumenta o grau de dificuldade do programa para executar a tarefa, bem como a sua validação, isto é, a determinação de que o programa traduz o que se pretende ou não. A complexidade pode ser reduzida através da modularização. Esta traduz-se na divisão de uma tarefa complexa num certo número de outras mais pequenas e independentes que depois se relacionam, a fim de que a respectiva recomposição gere o resultado pretendido inicialmente. A modularização é uma actividade intuitiva, cuja concretização não é óbvia, *a priori*.

O número de erros num programa depende, quer da complexidade, quer da sua dimensão. Uma maior dimensão de um programa requer um aumento do número de indivíduos envolvidos no seu desenvolvimento. Isto, por sua vez, gera uma maior possibilidade de ocorrência de erros, pois há uma multiplicação dos canais de comunicação: um para duas pessoas, três para três pessoas, seis

para quatro, etc. Brooks (1982) discute a questão da multiplicidade dos canais de comunicação, sugerindo que esta é uma das razões pela qual existe um limite aceitável de intervenientes no processo de desenvolvimento, o que origina impossibilidades na recuperação de atraso nos projectos, apenas pela mera soma algébrica de mais elementos ao dito processo. Uma das funções relevantes da gestão dos projectos é, precisamente, a diminuição da complexidade na comunicação, através da formação de equipas lideradas por um responsável, o elemento porta-voz do grupo.

O meio de comunicação é outro dos factores que pode gerar complexidade quando a dimensão aumenta. Saliente-se o exemplo das especificações funcionais, desenvolvidas através de uma linguagem natural escrita e descritiva com suporte manual, por oposição à utilização de ferramentas gráficas e diagramáticas com suporte informático. Sem dúvida que, no segundo caso, é possível garantir menor ambiguidade, mais consistência, maior fiabilidade e maior facilidade de actualização e revisão. Daqui o grande desenvolvimento que estas ferramentas têm alcançado na última década, embora a intangibilidade associada ao processo de especificação gere dificuldades particulares.

Os prazos de desenvolvimento dos sistemas de informação influenciam o custo do projecto, na medida em que este cresce proporcionalmente ao atraso. Por outro lado, atrasos significativos poderão causar o obsoletismo do projecto, ou de certas partes, antes de finalizado. Existe, pois, um equilíbrio delicado entre a possibilidade de obsoletismo e a tentação de se reverem, indiscriminada e repetidamente, as especificações do projecto. Projectos longos, em que as pessoas entram e saem e, em que, a documentação do projecto e das suas actividades é necessariamente uma actividade relevante, poderão, por vezes, desprezar este aspecto.

Os custos de desenvolvimento do *software* excedem quatro ou cinco vezes os custos do *hardware*, sendo a tendência para que este factor aumente. Como tal, as questões relacionadas com a eficiência e a produtividade assumem cada vez maior importância. Espera-se que o melhoramento da utilização de certas práticas, ferramentas e técnicas contribuam de uma forma positiva para estes aspectos.

Para além da utilização das técnicas e ferramentas, também a gestão dos projectos desempenha um papel relevante na eficiência e produtividade. Para isso contribui a elaboração de um plano do projecto. Este descreve, *grosso modo*, as actividades a implementar e os respectivos prazos de implementação. Contém ainda os recursos a afectar para que os objectivos do projecto, anteriormente referidos, sejam alcançados. Permite, por outro lado, a fixação de metas claras e objectivas, consoante os vários estágios do projecto, as quais facultarão a possibilidade de um eficaz controlo do progresso e da eficiência do desempenho.

2.2. Breve historial do desenvolvimento dos sistemas de informação

Nos anos 50, não estavam formalizadas metodologias para o desenvolvimento dos chamados sistemas de processamento de dados. Nessa altura, as aplicações científicas eram, também, uma componente relevante da computação. As aplicações comerciais eram escassas e estavam orientadas para o nível executivo/operacional da gestão. Exemplos típicos eram os relacionados com o processamento de dados, nomeadamente, copiar, ler, ordenar, guardar e verificar ficheiros de dados. Eram ainda efectuados alguns cálculos básicos que originavam relatórios sobre clientes e vendas, facturação automática e processamento de salários.

Basicamente, existiam dois tipos de agentes relacionados com o desenvolvimento dos sistemas de informação: utilizadores e programadores. Estes últimos eram os responsáveis pela implementação dos sistemas. Os projectos eram pontuais e localizados, difíceis de manter e actualizar, desprovidos de documentação (ou estando esta desactualizada), caros, chegavam sempre fora do prazo e nem sempre funcionavam bem, com excepção para as situações que os programadores conheciam muito bem. Os programadores possuíam um estatuto privilegiado, eram bem pagos, difíceis de substituir e eram os únicos a conhecer o trabalho que faziam, devido à ausência de documentação e de ferramentas de análise.

À medida que se caminhou para os anos 60 e 70 foram introduzidas novas funções: os analistas de sistemas e os operadores. Mais tarde a função de análise separou-se numa componente orientada para o negócio e noutra preponderantemente técnica.

De facto, um longo caminho foi percorrido no campo dos *automated systems*, isto é, dos sistemas desenvolvidos pelo homem que interagem com ou são controlados por um ou mais computadores. De seguida, enumeram-se alguns dos principais componentes destes sistemas que são comuns a todos: o *hardware* que é constituído pelos CPUs (unidades centrais de processamento), discos magnéticos, terminais, impressoras, fitas magnéticas, discos ópticos, etc; e o *software*, cujos exemplos são os sistemas operativos, os sistemas de bases de dados, os programas de controle de telecomunicações e os programas que implementam as funções requeridas pelos utilizadores. Também as pessoas integram os componentes dos *automated systems*, nomeadamente facultando-lhes *inputs*, utilizando os seus *outputs* e fornecendo as actividades de processamento manual requeridas. Outros exemplos são os dados, isto é, informação não interpretada e os procedimentos que são constituídos pelas políticas e instruções necessárias à operação dos sistemas.

Uma possível classificação dos diversos tipos de *automated systems* são os *batch*, os *on-line*, os *real-time*, os *decision-support* e os *knowledge-based systems*.

Nos *batch systems*, comuns nos anos 60 e 70, a informação era acedida sequencialmente, ao contrário dos *on-line* em que é possível o acesso aleatório. Os *on-line systems* (Yourdon, 1972) são os que aceitam os *inputs* directamente da área onde estes são criados e que enviam os *outputs* directamente para onde eles são requeridos.

Os *real-time systems* (Martin, 1967) podem definir-se como os sistemas que controlam algo na sua envolvente, através da recepção de dados, do seu processamento e do envio dos resultados com a rapidez suficiente para afectar a envolvente naquele instante. Alguns exemplos são os sistemas de controlo de processo, as *automated teller machines* (ATM), os sistemas de monitoria de pacientes, os sistemas de controlo de mísseis, entre outros.

Os *decision-support systems* não tomam decisões sozinhos, mas ajudam os gestores a tomar decisões de forma suportada. Surgem na sequência e completam os *transaction-processing systems*, sistemas operacionais tradicionalmente ilustrados pelos sistemas de processamento de salários, da contabilidade, etc. Podem aqui incluir-se, para além dos sistemas de informação de apoio à gestão, também as folhas de cálculo, as bases de dados, os sistemas de análise estatística, os sistemas de previsão, etc. Numa fase posterior surgem, nesta classe, os *strategic planning systems* que facultam aos gestores de topo uma perspectiva abrangente e de aconselhamento geral acerca da natureza do mercado, das preferências dos consumidores, do comportamento da concorrência, etc.

Finalmente, têm-se os *knowledge-based systems* que aparecem associados à inteligência artificial. Segundo Elaine Rich (Rich, 1984), o objectivo dos cientistas da computação, a trabalhar no campo da inteligência artificial, é a produção de programas que imitem o desempenho humano numa grande variedade de tarefas inteligentes. Um *expert system* é uma espécie de *knowledge-based system* (Feigenbaum e McCorduck, 1983). É um programa de computador que tem imbutidos conhecimento e capacidade que lhe permitem operar ao nível de um *expert*. O *expert system* faculta um apoio intelectual de alto nível ao *expert* humano, o que explica o seu outro nome: assistente inteligente.

2.3. A metodologia convencional de análise de sistemas

Definida que está a nossa área de interesse — a dos *decision-support systems* —, importa agora ver o que se entende por sistema de informação, antes de se introduzir o tema do parágrafo.

Buckingham (1987) define sistema de informação como um sistema que combina, guarda, processa e fornece a informação relevante para uma organização (ou à sociedade), de tal modo que a informação é acessível e útil a todos os que desejarem utilizá-la, incluindo gestores, *staff*, clientes e cidadãos. Um sistema de

informação representa um sistema de actividade humana (social) que pode ou não envolver o uso de computadores.

Avison (1994) salienta a diferença entre sistemas de informação e sistemas de processamento de dados, concluindo que estes implicam, à partida, a necessidade de diferenças nas respectivas metodologias de desenvolvimento. Os sistemas de informação deverão reflectir as necessidades dos gestores em informação, incluindo todos os sistemas de processamento de dados da organização e acrescentando-lhes uma componente de integração. A metodologia que se vai descrever de seguida é uma das formas que toma a recomendada pelo *National Computing Centre* do Reino Unido, sendo conhecida pelas designações *análise de sistemas convencional*, *análise de sistemas tradicional*, *ciclo de vida do desenvolvimento de sistemas* ou *modelo da queda de água*. Representa um processo iterativo de abordagem em que cada estágio é seguido por uma revisão, cujo objectivo é verificar a sua adequabilidade. Caso isto não suceda, o processo poderá recommençar com um novo estudo prévio.

A Análise de Sistemas Convencional introduziu grandes melhoramentos nos processos de desenvolvimento de sistemas anteriormente usados, nomeadamente as questões inerentes à própria definição de metodologia, isto é: uma abordagem faseada, desde o estudo prévio até à fase de manutenção, passando pela investigação, análise, projecto e implementação; uma série de técnicas, tais como as relativas à análise de custos-benefícios, para além das que adiante se descreverão; algumas ferramentas, tais como as associadas à gestão de projectos; procedimentos de treino devidamente enquadrados no contexto da análise; e, finalmente, uma filosofia implícita de que os computadores facultam boas soluções para problemas administrativos.

2.3.1. Fases da Análise de Sistemas Convencional

2.3.1.1. Estudo prévio

Proporciona uma perspectiva do sistema actual, dos seus problemas e avança soluções alternativas que podem englobar componentes manuais e informatizadas. Inclui uma descrição dos aspectos técnicos, humanos e económicos do desenvolvimento e operação do sistema existente, sumariados num relatório a apresentar à administração que deverá ainda recomendar uma solução.

2.3.1.2. Investigação do sistema

Se a administração aprovar o estudo e der o seu acordo para que ele prossiga, deverá passar-se à fase de investigação da área de aplicação. O objectivo é a deter-

minação dos requisitos funcionais do sistema, das restrições, das condições de excepção, dos tipos de dados e respectivos volumes e de eventuais problemas dos métodos de trabalho empregues.

Para além da utilização de questionários, técnicas de entrevista e observação, esta fase é documentada através de fluxogramas, organigramas, especificações dos documentos do sistema manual, grelhas de análise, actas de reuniões e notas avulso dos analistas.

2.3.1.3. Análise de sistemas

Com base na informação recolhida, o analista deverá colocar, entre outras, algumas questões sobre a razão de ser dos problemas, a adopção de certos métodos de trabalho, potenciais alternativas a estes e as taxas de crescimento prováveis para diversos tipos de dados.

2.3.1.4. Projecto de Sistemas

O projecto tradicional caracteriza-se por dois tipos de condicionamentos: por um lado, as restrições impostas pela solução esboçada, durante a fase de estudo prévio e, por outro, as limitações resultantes do sistema manual que, na maior parte das vezes, originam uma solução informatizada similar à manual, mas recorrendo ao poder de processamento proporcionado pela ferramenta informática.

Os principais aspectos tratados, nesta fase, referem-se a:

- *definição dos dados requeridos para alimentação* do sistema e respectivo processo de aquisição e entrada no sistema;
- *especificação dos outputs* gerados;
- *especificação do processamento dos inputs nos outputs*;
- *definição da estrutura do sistema* informático;
- *definição da estrutura de ficheiros* manuais e informáticos;
- *especificação dos ficheiros*, nomeadamente o suporte (disco ou fita), a organização (acesso sequencial ou aleatório), o dimensionamento, os procedimentos de *back-up* (cópias de segurança) e a especificação dos registos (nome, descrição, formato e gamas de valores);
- *especificação dos procedimentos de segurança*;
- *planeamento dos testes requeridos e implementação*.

As técnicas mais utilizadas são os gráficos de *input-output*, os impressos de especificação dos ficheiros, dos *inputs* e dos *outputs*, as grelhas de análise que relacionam ficheiros e programas e os fluxogramas que exibem os fluxos de procedimentos.

O projecto do sistema inclui o projecto e codificação dos programas, actividades desenvolvidas pelos programadores. Isto representa uma separação das tarefas de análise e de projecto e, consequentemente, um melhoramento relativamente à situação em que ambas as funções estavam conjugadas. Os testes dos programas poderão ser incluídos na fase seguinte.

2.3.1.5. Implementação

Nesta fase, incluem-se todos os procedimentos conducentes à implantação do novo sistema, nomeadamente a prática de utilização, o treino e formação dos recursos humanos, a documentação do sistema, a validação, verificação e o arranque operativo.

A operação dos novos sistemas era, normalmente, iniciada faseadamente e em paralelo com os sistemas antigos, procedimentos conhecidos por *arranque faseado* e *arranque piloto* do sistema.

2.3.1.6. Revisão e manutenção

O último estágio desta metodologia, a manutenção, ocorre já quando o sistema está operacional. Resulta da necessidade de se efectuarem ajustamentos e mudanças que garantam a sua eficiente operação. Por outro lado, deverá ocorrer, em todos os estágios, uma revisão que garanta a conformidade do sistema face aos requisitos estabelecidos durante o estudo prévio. Quer os resultados destas revisões, quer eventuais desvios face às opções previamente aprovadas deverão ser analisados, discutidos e eventualmente justificados num relatório final que poderá resultar da compilação de vários parciais.

2.3.2. Críticas à perspectiva convencional

Avison (1994) resume as principais objecções à perspectiva tradicional de desenvolver sistemas de informação, nomeadamente:

- *falha na conformidade com as necessidades da gestão*, devido à ênfase no processamento de dados, *de per si*, isto é, na vertente operacional e rotineira dos sistemas de informação, ignorando os requisitos dos níveis tático e estratégico da gestão;
- *projecto pouco ambicioso*, implicando uma mera substituição, com automação dos sistemas manuais, sem preocupações de conceber algoritmos e procedimentos que explorem as capacidades da nova ferramenta;

- *dificuldade de adaptação das ferramentas de desenvolvimento de sistemas* de informação às constantes necessidades de actualização, geradas pela dinâmica de mudança que ocorre na envolvente do negócio;
- *sistemas projectados de modo orientado para os outputs*, implicando que todo o desenvolvimento e também os *inputs* sejam condicionados pelos *outputs*, o que, em caso de mudança destes, origina um enorme volume de modificações e grandes atrasos no projecto dos sistemas;
- *insatisfação dos utilizadores* que, por vezes, gera rejeição dos sistemas por parte destes, aquando da implementação, talvez por, somente muito tardiamente, se aperceberem do impacto das suas opções; estas decisões, assumidas muito cedo, mostram-se algumas vezes inadequadas e de difícil adaptação, devido à inflexibilidade dos procedimentos de desenvolvimento;
- *procedimentos de desenvolvimento orientados para os especialistas*, mostrando-se pouco adequados ao entendimento e utilização por parte dos utilizadores, gerando a desilusão e retracção destes que, assim, se sentem pouco motivados a colaborar na produção de sistemas que sejam conformes com as suas necessidades; daqui resultam, por vezes, sistemas informais que, funcionando em paralelo, contribuem para o obsoletismo precoce, ineficácia e abandono dos sistemas «oficiais»;
- *os principais problemas com a documentação* são de dois tipos: por um lado, a sua orientação para questões técnicas e para os especialistas e, por outro, a relutância dos técnicos, em documentarem os sistemas e o seu funcionamento, causam dificuldades na actualização, originando o desinteresse dos utilizadores;
- *sistemas incompletos* que ignoram as excepções, ou cujo tratamento foi esquecido, durante a fase de análise, causam habitualmente problemas, aquando da entrada em funcionamento; a solução passa, muitas vezes, por completar as actividades de processamento estruturadas, estáveis, rotineiras e, consequentemente, automatizadas e facilmente automatizáveis com procedimentos manuais;
- *a fila de espera para o desenvolvimento de aplicações* pode causar dois tipos de inconvenientes: ou os utilizadores ficam desmotivados e desistem, ou os sistemas demoram tanto tempo que entretanto algo se modificou, alterando as necessidades ou criando mesmo situações de obsoletismo;
- *a sobrecarga das actividades de manutenção* origina, muitas vezes, a tentação de soluções rápidas e fáceis, mas menos pensadas e estruturadas; por outro lado, com cerca de 60 a 80% dos recursos dos departamentos de desenvolvimento de sistemas afectados à manutenção, é natural que se criem as filas de espera anteriormente referidas.

Algumas constatações preocupantes são que, mesmo hoje em dia, existem ainda muitos sistemas que são desenvolvidos sem qualquer metodologia,

seguindo perspectivas *bottom up* (abordagem pontual, com preocupação predominante no detalhe do problema, de modo mais ou menos isolado do contexto) e com ênfase exclusiva no empirismo e no improviso. Em particular, esta atitude encontra-se frequentemente no campo da microinformática, em que os erros cometidos são muito similares aos ocorridos, nos anos 60, com as aplicações para *mainframes*.

Finalmente, verifica-se também que, mesmo seguindo metodologias alternativas, os grandes problemas associados ao desenvolvimento de sistemas persistem, isto é: o não cumprimento de prazos, a deficiente manutenção dos programas, o recrutamento e retenção dos recursos humanos e a insatisfação geral dos utilizadores.

2.4. Outras metodologias de análise de sistemas

2.4.1. *Software Engineering* (SE)

Hoje em dia é reconhecido que a metodologia *Software Engineering* contribuiu de modo significativo para o melhoramento do projecto do *software*. A fase de projecto dos métodos convencionais baseava-se em fluxogramas, o que originava programas pouco estruturados e não modulares, com um sem número de instruções *GOTO*. Este código, chamado *spaghetti*, era extremamente difícil de manter e actualizar, devido ao elevado número de impactos cruzados resultantes das modificações introduzidas. Por outro lado, quer a imposição de soluções à partida, durante o estudo prévio, quer o peso atribuído aos sistemas antigos, condicionavam a utilidade do processo de análise e, portanto, da especificação de requisitos dos utilizadores. A disciplina de abordagem introduzida pela *Software Engineering* também se manifestou na melhoria substancial da documentação. De facto, a manutenção que envolve a actualização dos programas, cujos requisitos mudaram, bem como a correcção dos programas que não funcionam correctamente, foi facilitada pelo melhoramento da documentação e do projecto introduzidos pela SE. Estes dois melhoramentos aumentam significativamente a produtividade, através da poupança introduzida nos testes e manutenção dos programas que ultrapassa largamente o sobrecusto do projecto, inerente à utilização da *software engineering*.

Por outro lado, a análise *top down*, característica da SE, dá prioridade ao projecto modular das interfaces: é mais fácil corrigir um problema bem localizado num módulo, do que juntar programas e/ou sistemas, depois destes terem sido desenvolvidos de forma independente.

Finalmente, o processo de desenvolvimento faseado e iterativo promove o aumento da fiabilidade e a garantia de qualidade do trabalho realizado, permitindo

o procedimento conhecido por validação que confronta o resultado do trabalho produzido com as necessidades expressas pelo utilizador, inferindo as correcções necessárias, antes da passagem ao estágio posterior da metodologia. Os diversos estágios da metodologia passam de seguida a ser analisados.

2.4.1.1. Análise e especificação das necessidades do utilizador

Âmbito

Durante este estágio, a ênfase é colocada na função do *software*, sendo devidamente testada a consistência interna dos requisitos. Também denominada análise de sistemas, no contexto dos sistemas de processamento de dados, a análise difere essencialmente da especificação, pela linguagem em que é expressa. Esta, é a da parte contratante, o cliente, enquanto que o documento de especificação do *software* é traduzido para a linguagem do subcontratado, a *software house*. O documento-síntese da análise das necessidades deverá ampliar, interpretar e codificar estes requisitos, sendo claro, isto é, não ambíguo, consistente e não contraditório. Muitas vezes, estas duas fases encontram-se fundidas num só estágio, simplesmente denominado especificação de necessidades.

A especificação dos requisitos deverá incluir três tipos de informação: uma especificação funcional, as restrições e os objectivos.

Especificação funcional

A especificação funcional do *software* descreve com precisão «o que» o sistema deverá fazer, sem se preocupar com «o modo como» isso deverá ser alcançado, segundo o que foi pensado na fase de projecto. A funcionalidade do *software* deverá contemplar os seguintes aspectos:

- *especificação dos dados* que são necessários ao correcto funcionamento do sistema;
- *especificação do hardware* requerido;
- *especificação do nível de desempenho*, isto é, de aspectos tais como tempos de resposta e tempo médio entre falhas;
- *especificação dos testes* que asseguram a conformidade do *software* com a especificação;
- *especificação do treino requerido* pelos recursos humanos do contratante para operar e manter o *software*.

Restrições ou requisitos não-funcionais e objectivos

Por vezes, existem condicionamentos do comportamento do sistema que surgem na fase de especificação por força da vontade do cliente. Estes aspectos limitam a liberdade de escolha dos projectistas. A fim de se manter a especificação funcional tão livre quanto possível destes condicionalismos, deverão ser agrupados numa rubrica denominada restrições.

Citam-se, como exemplos de restrições, a imposição de uma determinada linguagem, os prazos de desenvolvimento do projecto, as normas para os documentos, as medidas do desempenho, as verbas a afectar ao projecto, o respeito pela cultura, procedimentos e *standards* da empresa, a obrigatoriedade de recorrer às rotinas do sistema operativo para determinadas operações, entre outras.

Os objectivos são os aspectos da especificação das necessidades que guiam o projectista, orientando-o face à liberdade de escolha existente.

Como exemplo de objectivos citam-se as opções de “performance” que o cliente faz, isto é, os tempos de acesso ao disco, os tempos de processamento, os tempos de resposta a sinais externos e o nível de fiabilidade pretendido, em detrimento de outros aspectos, como sejam, os custos e os prazos de desenvolvimento. São, pois, indicações que permitem ao projectista fazer as melhores opções face às necessidades do cliente.

Documento de especificação do software

O documento de especificação do *software* é uma peça fundamental da gestão, do desenvolvimento e da manutenção dos sistemas de informação, dado que permite registar, de forma inequívoca, as necessidades do cliente. Assim, é possível, *a posteriori*, quer monitorar o progresso do projecto, quer avaliar o nível de satisfação dos requisitos estabelecidos, quer progredir para os estágios seguintes de forma orientada, quer ainda suportar devidamente as intervenções de manutenção.

Este documento deverá estabelecer, essencialmente, o que se pretende que o *software* execute e alcance, através da descrição funcional, da explicitação das restrições e dos objectivos. Deverá ser fácil de mudar, embora sejam indesejáveis actualizações depois da especificação ter sido devidamente aceite e aprovada pelo cliente.

A lista seguinte representa uma estrutura possível para o dito documento, de acordo com Thomas (1994).

Secção 1 — Introdução

- 1.1 Identificação
- 1.2 Introdução
- 1.3 Resumo funcional
- 1.4 Hipóteses e restrições

Secção 2 — Hardware

- 2.1 Descrição do *hardware*
- 2.2 Descrição das interfaces

Secção 3 — Requisitos funcionais

Secção 4 — Requisitos não-funcionais

Secção 5 — Garantia da qualidade

- 5.1 Requisitos para a qualidade
- 5.2 Procedimentos de teste do *software*
- 5.3 Procedimentos de validação do *software*
- 5.4 Critérios de aceitação

Secção 6 — Especificação da manutenção

- 6.1 Hipóteses de fundamentação do sistema
- 6.2 Antecipação de mudanças futuras
- 6.3 Preparação de relatórios e mensagens de erro

Secção 7 — Especificação do treino

- 7.1 Treino dos operadores
- 7.2 Treino dos operadores do *software*
- 7.3 Treino dos operadores de *hardware*

Secção 8 — Índices

- 8.1 Índice alfabético
- 8.2 Índice dos capítulos
- 8.3 Índice funcional

Apêndice A — Acrónimos e abreviaturas

Apêndice B — *Standards* de desenvolvimento de *software* da *software house*

Já Sommerville (1994) apresenta uma estrutura ligeiramente diferente, em que passa para apêndice todas as questões de detalhe sobre especificação funcional, não-funcional, de *hardware* e de base de dados.

Validação da especificação de necessidades

A especificação de um *software* deve ser consistente, completa, realística e evitar a ambiguidade. A operação de validação também deverá verificar estas características, para além de confirmar a especificação do produto certo face às necessidades reais saídas da análise. Dado que estas são, muitas vezes, expressas em linguagem natural, existem problemas de ambiguidade e imprecisão que dificultam, consideravelmente, a garantia de conformidade da especificação com os requisitos resultantes da análise.

Por vezes, a melhor forma de efectuar a validação é através de um protótipo, o que, garantindo melhores resultados, encarece substancialmente a operação.

Técnicas

Durante a análise, as técnicas de recolha de dados e/ou informação mais populares são os questionários, as entrevistas, a observação directa, as pesquisas em arquivos manuais ou electrónicos, a linguagem natural, a amostragem e outras técnicas estatísticas usadas pela organização e métodos.

Por outro lado, a especificação tende a ser expressa em notações gráficas, embora estas sejam completadas por outras, nomeadamente as notações matemáticas, as descrições em linguagem estruturada e as tabelas de decisão. Como exemplo das notações gráficas correntemente mais populares, quer no processo de análise, quer no de especificação, encontram-se os diagramas de fluxo de dados (DFD), os de entidades-relações (E-R) e os de transição entre estados.

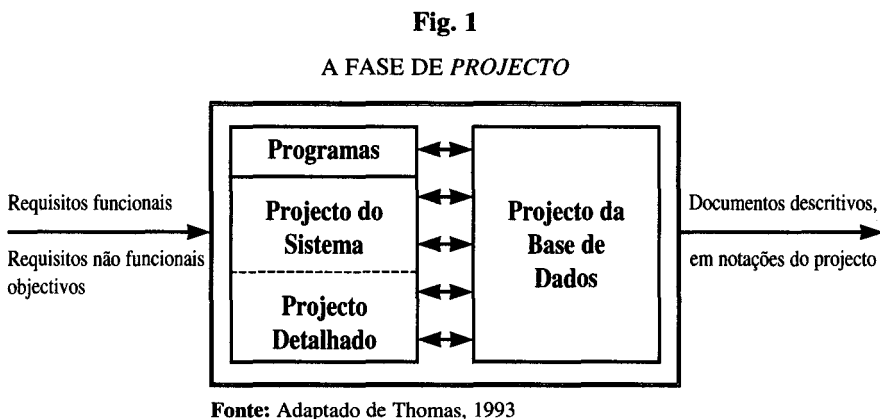
O primeiro passo na produção de uma especificação da funcionalidade do *software* é a conversão dos requisitos do cliente num diagrama de fluxo de dados. Esta operação transforma a especificação funcional numa forma adequada para o projectista. Pela sua importância, é relevante que se tenham algumas considerações sobre as características da técnica DFD. De Marco (1978) definiu as propriedades que uma especificação funcional deve possuir e demonstrou que todas elas são satisfeitas pelos DFD, como se segue:

- *deve ser gráfica*, uma vez que a visualização de figuras simplifica a interpretação;
- *deve ser modular*, mantendo as diferentes funções separadas, a fim de facilitar o desenvolvimento e a manutenção;
- *deve ser fácil de manter*, minimizando-se a redundância e a duplicação e, consequentemente, evitando a repetição de uma mesma correcção;
- *deve ser lógica e não física*, o que evita modificações desnecessárias devido, por exemplo, a mudanças de *hardware*;
- *deve ser precisa, concisa e legível*.

2.4.1.2. Projecto

O objectivo da fase de projecto é o desenvolvimento da especificação até ao ponto em que possa ser implementada como um programa. A principal divisão, entre a especificação e o projecto, é que este tem como objecto do seu interesse definir *como* alcançar o *que* é requerido na especificação.

Na **Figura 1** representam-se os dois aspectos com que o projecto essencialmente se preocupa: a concepção dos programas e a dos dados que os alimentam. Representam-se igualmente os *inputs* e *outputs* da fase de projecto.



Os *outputs* da fase de projecto, expressos nas notações respectivas, são os seguintes:

- *repartição da especificação global em termos de hardware, software existente e software a implementar;*
- *projecto do sistema, isto é, modularização do software;*
- *projecto detalhado, isto é, desenvolvimento dos algoritmos de cada módulo;*
- *projecto da estrutura de dados necessários aos algoritmos concebidos;*
- *planeamento dos procedimentos de teste do software.*

Projecto do sistema

A actividade subjacente ao método *projecto estruturado do sistema* consiste em transformar a especificação funcional num modelo hierárquico, constituído por uma série de procedimentos e funções. Esta tarefa gera ainda a descrição das interfaces respectivas.

De facto, a modularização divide uma grande tarefa em subtarefas, executadas por módulos que são chamados por um programa principal. Este recebe alguns *inputs* e gera *outputs* de e para a sua envolvente. Os módulos, para além da possibilidade de interacção com a envolvente, interfaceiam igualmente entre si.

Segundo Thomas (1993), a técnica usada para transformar a notação DFD (diagrama de fluxo de dados), proveniente da especificação na notação *gráfico estru-*

turado, associada à fase de projecto, é a *transformational analysis*, apropriada para programas até cerca de 5 000 instruções.

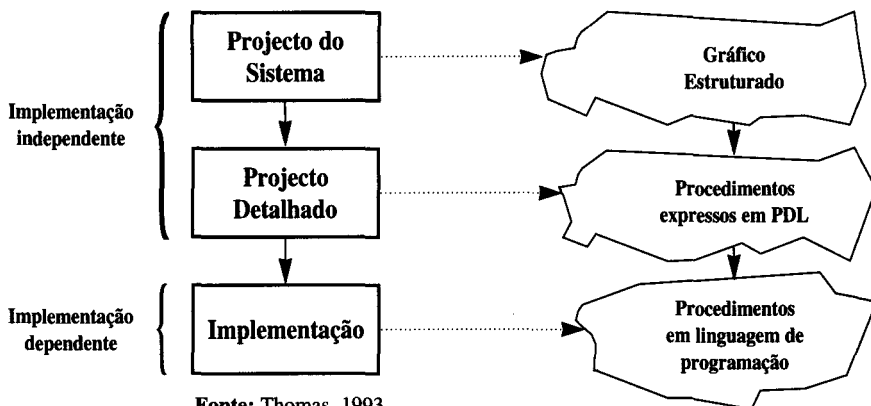
Projecto detalhado

Terminada a modularização, fica completado o projecto do sistema. Passa-se então ao projecto detalhado, em que é concebida a lógica interna de cada módulo, a partir dos gráficos estruturados. O método utilizado é conhecido por projecto *top down*, ou por decomposição funcional, ou ainda por *stepwise refinement*. Uma das notações mais generalizadas é a *program design language* (PDL) que se assemelha a uma linguagem de programação sem especificidades, pelo que pode ser facilmente implementada pelos programadores, *a posteriori*, numa diversidade de linguagens, por exemplo: *Pascal*, *Ada* ou *Modula 2*.

As PDL apresentam como vantagens uma maior facilidade de entendimento da lógica do que as linguagens de programação e uma independência face a estas que permite, quer uma fácil mudança, em caso de necessidade, quer a ausência das restrições inerentes às linguagens.

Fig. 2

RELAÇÕES ENTRE PROJECTO E IMPLEMENTAÇÃO



Fonte: Thomas, 1993

2.4.1.3. Implementação

A implementação, no âmbito da *Software Engineering*, consta da conversão do projecto detalhado, expresso numa PDL, para as unidades de programa, tais como

os procedimentos, as subrotinas e funções, consoante mostrado na **Figura 2**. Limita-se, portanto, à simples tradução para instruções da linguagem de programação escolhida da notação PDL, operação que é efectuada pelos programadores e é denominada codificação.

Quando se decompõe um procedimento em instruções PDL, encontram-se os seguintes *tipos de instruções*:

- *entrada e saída de dados, atribuição, definição de variáveis e chamada de outros procedimentos;*
- *selecção, tais como, IF-THEN-ELSE e CASE-OTHERWISE;*
- *repetição, tais como os ciclos FOR, WHILE e REPEAT.*

A verificação do projecto detalhado pode fazer-se através da inserção de asserções no código. As pré-asserções descrevem, em lógica formal, as condições que se deverão verificar antes de um programa, procedimento ou troço de código ser executado, enquanto que as pós-asserções descrevem as condições a verificar depois da execução.

2.4.1.4. *Manutenção*

Âmbito

O estágio de manutenção tem por objecto os melhoramentos a introduzir no *software*, depois de terminado o seu ciclo de desenvolvimento. A sua importância, para além de garantir que se mantêm actualizadas as características funcionais do sistema, recai também em aspectos de natureza económica. De facto, nas organizações, a maior parte dos recursos afectos ao desenvolvimento de sistemas estão envolvidos em operações de manutenção.

Lehman e Belady (1985) dedicaram-se ao estudo da evolução da dinâmica dos programas, tendo chegado, entre outras, às seguintes conclusões, com base em observações do crescimento e evolução de um certo número de grandes sistemas:

- *a manutenção como processo de correcção de falhas e de adaptação das mudanças, ocorridas na especificação, é um processo inevitável;*
- *a degradação da estrutura do sistema ocorre à medida que este vai mudando, pelo que há necessidade de operações explícitas de reestruturação para reverter este processo, devendo ser previstos os custos adicionais respectivos;*
- *os sistemas de grande porte possuem uma dinâmica própria que se reflecte nas necessidades e também em certas limitações, inerentes ao processo de*

manutenção; essa dinâmica é estabelecida nos primeiros estágios do seu ciclo de desenvolvimento, implicando, quer um número máximo de modificações, quer um prazo máximo para a sua ocorrência, a partir dos quais a funcionalidade e a fiabilidade do sistema se degradam irreversivelmente.

Custos de manutenção

De acordo com Sommerville (1994), os custos de manutenção são difíceis de estimar, sendo-o, normalmente, por defeito, aquando do desenvolvimento e implementação do sistema. Estes custos são, de longe, os mais elevados em que se incorre, variando entre duas a quatro vezes a sua proporção para os custos de desenvolvimento.

Com vista à redução dos custos de manutenção, é eficaz o investimento de mais esforço e tempo, aquando do desenvolvimento de um sistema.

A **Figura 3** expressa, de acordo com Sommerville (1994), os principais factores de dependência dos custos de manutenção.

Fig. 3

FACTORES DE DEPENDÊNCIA DOS CUSTOS DE MANUTENÇÃO

Factores de cariz não técnico	Factores de cariz técnico
Domínio de aplicação Estabilidade do <i>staff</i> Idade dos programas Envolvente externa Estabilidade do <i>hardware</i>	Independência modular Linguagem de programação Estilo de programação Validação do programa Documentação

2.4.1.5. *Garantia da Qualidade*

A garantia da qualidade assegura que, no fim do ciclo de desenvolvimento, o *software* possua o nível de qualidade aceitável. Para isso, há que monitorar continuamente o processo de desenvolvimento através do ciclo respectivo.

Há quatro aspectos fundamentais no processo de garantia da qualidade: a verificação, a validação, o teste e a robustez do *software*.

O processo de verificação assegura que o *output* de uma fase representa a correcta conversão do *input* dessa fase e, como tal, averigua a correcção do processo de transformação do *input* no *output*.

A validação assegura que o *output* de cada fase do ciclo de vida está conforme com os requisitos do utilizador.

O processo de teste assegura a validação do *software* completo, ou das suas partes, de modo isolado. Existem duas possibilidades de teste. Por um lado, o *software* pode conter código para verificação de erros, o que ocasiona a diminuição da velocidade de execução e maiores necessidades de espaço de armazenagem em disco. Em alternativa, têm-se técnicas de teste que verificam a execução de todas as partes do programa, através de um conjunto especialmente construído de dados de teste e da subsequente análise dos *outputs* gerados. Estes testes de comportamento são sequenciais, consumindo um tempo considerável. Uma consideração relevante acerca do processo de teste é que este apenas acusa a presença de erros, não garantindo a sua ausência. Por outro lado, o processo de *debugging* preocupa-se com a localização e remoção dos erros detectados.

O conceito de robustez parte do princípio que todos os programas contêm alguns erros. Então, é necessário garantir que o *software* reaja sempre de modo fiável para o utilizador.

Sommerville (1994) afirma que o *software* nunca deverá produzir *outputs* incorrectos, independentemente dos *inputs*; não deverá poder ser corrompido, deverá gerar acções úteis e com significado, perante situações inesperadas e somente deverá falhar completamente quando o progresso for absolutamente impossível.

Aplicação da Total Quality Management (TQM) ao desenvolvimento de software

Heal (1993) propõe a aplicação do conceito TQM através da sua vertente cliente-fornecedor. Isto implica que qualquer das fases da cadeia de desenvolvimento se coloque na posição de cliente, que não aceita que lhe sejam entregues produtos defeituosos, e que também se coloque na situação de fornecedor, que não aprovisiona os seus clientes com produto defeituoso.

A **Figura 4** mostra o ciclo de desenvolvimento de *software* como um processo *top down* — a análise e o projecto —, seguido de um processo *bottom up* — a implementação. Exibe ainda o conjunto estabelecido de testes intermédios, entre os vários estágios do ciclo que possibilitam a aplicação da filosofia TQM. Podem, portanto, identificar-se, quer as diversas relações cliente-fornecedor (internas e externas), quer os respectivos testes associados ao processo de garantia da qualidade total.

As cinco etapas do processo de teste expressas na **Figura 4** são as seguintes:

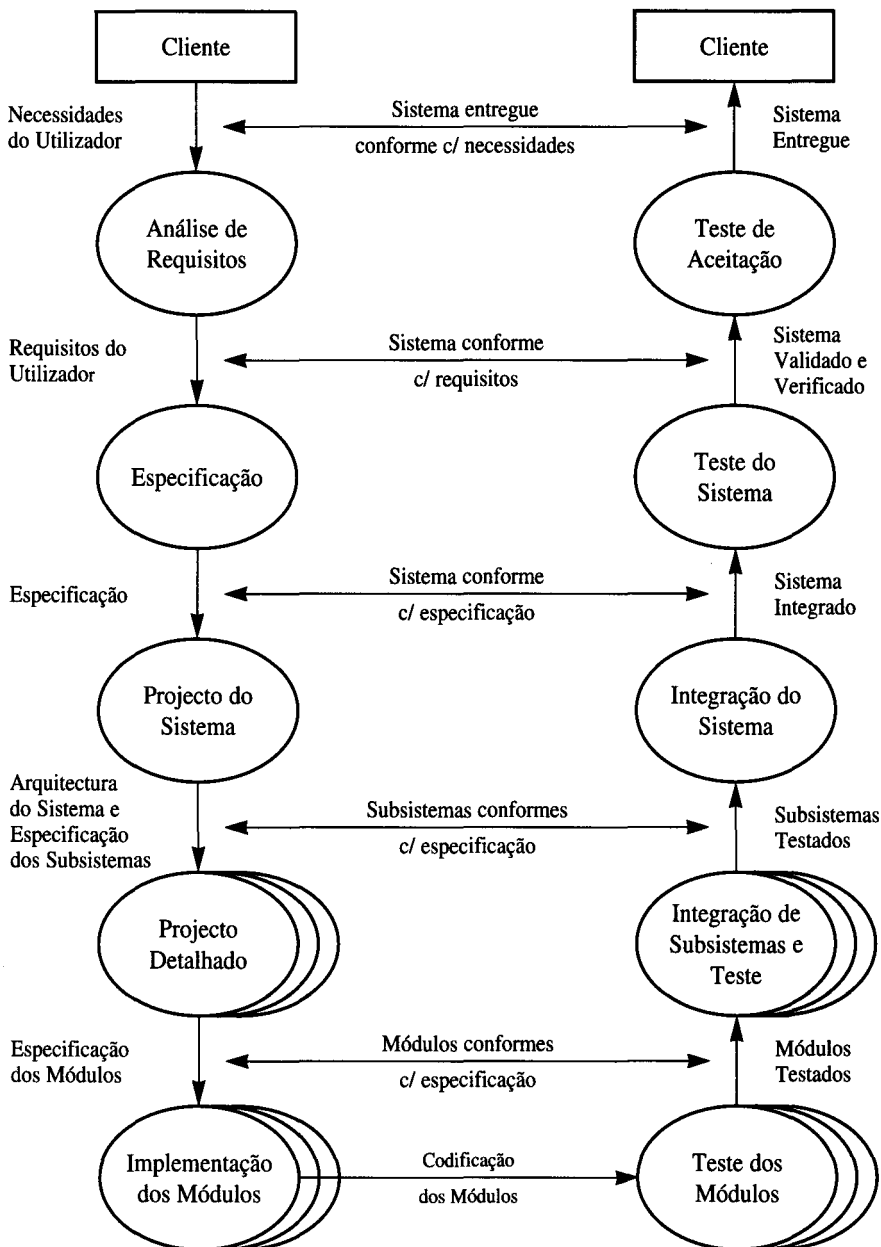
- o *teste de unidades* testa um procedimento ou função de modo isolado; inclui análise estrutural — estática e dinâmica —, no caso de unidades críticas ou contendo código complexo;
- o *teste de módulos* ocorre depois do das unidades e envolve o teste de cada unidade em conjunção com outro do módulo;
- o *teste de integração* é o processo de teste de uma versão de um *software* parcialmente construída; envolve a verificação de que as unidades, os módulos e os subsistemas, testados individualmente, de facto interfaceiam correctamente com o sistema em que vão ser integrados;
- o *teste do sistema* consiste em testes, demonstrações e análises que verifiquem se o sistema completo funciona de acordo com os requisitos especificados; deve ser levado a cabo num ambiente de *hardware* e de *software* o mais próximo possível dos do cliente, mas é regido pelas premissas do fornecedor;
- o *teste de aceitação* é um subconjunto do teste do sistema, escolhido pelo cliente; é efectuado sobre o sistema completo, pelo pessoal da qualidade em conjunto com o cliente, no ambiente operacional; tem como objectivo verificar que o código final está conforme com a especificação, pelo que, em caso de falha, todos os testes deverão ser repetidos no *software* corrigido.

Antes de se finalizar este parágrafo, é importante salientar que a garantia da qualidade é mais do que verificação e validação, incluindo também aspectos como a manutenção, fiabilidade, a amigabilidade e a portabilidade, entre outros. Os atributos relevantes para um dado projecto deverão ser explicitamente mencionados no plano de qualidade do *software*, devendo ainda ser fixado o modo da sua avaliação.

A fixação de padrões é importante para a garantia da qualidade que, em resumo, é a actividade de verificar a conformidade do processo e do produto face a estes *standards*. Infelizmente, a ausência de um processo de avaliação quantitativa fiável é ainda uma realidade.

Fig. 4

O CICLO DE DESENVOLVIMENTO DO SOFTWARE DO PONTO DE VISTA DO TESTE



Fonte: Heal, 1993

2.4.2. Protótipos e programação exploratória

Existem outros procedimentos de desenvolvimento de sistemas que não evoluem de forma ordenada, desde os requisitos do utilizador até à codificação. No entanto, alguns destes procedimentos podem ser mais encarados do ponto de vista de complementaridade do que do ponto de vista alternativo, surgindo, essencialmente, devido à dificuldade em explicitar as necessidades dos utilizadores. De seguida, enumeram-se algumas das principais razões justificativas destas dificuldades:

- *cliente especialista no domínio de aplicação*, ignorando, contudo, as diversas possibilidades tecnológicas ao seu dispor;
- *problemas de comunicação* entre os especialistas e os utilizadores;
- *dificuldades de expressão* de forma consistente e completa em linguagem natural;
- *dificuldades de previsão do impacto provocado pelo uso do computador* na operação e dos consequentes requisitos assim surgidos;
- *dificuldade de previsão de novos requisitos* associados a novas estratégias e formas de gestão (por exemplo, JIT), normalmente associadas ao processo de mudança em curso;
- *especificação de alguns aspectos num plano abstracto*, sem possibilidade de teste da correcção das decisões;
- *dificuldades de visualizar esquecimentos*.

2.4.2.1. Programação exploratória

No caso da programação exploratória aproveita-se uma determinada perspectiva da solução do problema para se desenvolver um sistema incompleto, abordando a maior parte dos aspectos principais, mas ignorando questões de robustez, eficiência e elegância. Este protótipo será depois modificado, completado e estendido de modo a comportar as partes que faltam e a colmatar as falhas encontradas. Como não há especificação do *software*, não é possível validá-lo face ao dito, podendo ainda ocorrer problemas devidos à ausência de planeamento de custos e a outros de carácter legal, caso o cliente não se sinta satisfeito. Poderão ainda ocorrer problemas de carácter técnico, devido ao desenvolvimento do sistema por mudança contínua, o que causa a degradação da estrutura, consoante anteriormente referido (ver § 2.4.1.4.).

2.4.2.2. Protótipos «de deitar fora»

Os protótipos «de deitar fora» têm por objectivo ajudar somente a melhorar o entendimento dos requisitos, sem fazer parte do produto final a entregar. Por isso,

são irrelevantes as preocupações com a estrutura do programa. Poderão ainda ignorar-se aspectos como o *hardware*, a velocidade de execução, a dimensão do código e a linguagem de programação, dados os objectivos do produto. Também um grande número dos requisitos funcionais serão omitidos ou bastante simplificados. Por exemplo, os relatórios que não levantem grandes dúvidas deverão ser ignorados.

As técnicas a utilizar deverão ser simples, rápidas e baratas, mesmo que isso implique penalizações, por exemplo, na velocidade de execução, dado que o protótipo não será aproveitado para produto final. Como exemplos, têm-se as linguagens de 4.^a geração (4GL) e o *Prolog*.

Se, por um lado, o protótipo deve clarificar de forma inequívoca os requisitos do utilizador que constituem o ponto de partida do sistema convencional, por outro, deverá ser entendido que o sistema final será necessariamente diferente, uma vez que conterà as considerações não-funcionais propositadamente ignoradas.

Deverá ainda evitar-se que o protótipo, ou a forma como está construído (por exemplo, a lentidão de uma dada função), influencie o utilizador na sua análise, de modo a que este ajuste o seu comportamento a características meramente conjunturais do protótipo.

Sommerville (1994) explicita ainda os principais estágios no desenvolvimento de protótipos, a saber:

- *fixação dos objectivos do protótipo*: deverão ser claros, de modo a que quem avalie saiba o que está a avaliar; exemplos de objectivos podem ser validar os requisitos funcionais, demonstrar a exequibilidade de dado aspecto, entre outros;
- *seleccionar funções, restrições e objectivos a considerar*: deverá definir-se o que incluir e o que deixar fora do protótipo, pois isso tem implicações relevantes nos custos ;
- *desenvolver o protótipo*;
- *avaliar o protótipo*: Ince e Hekmatpour (1987) consideram este estágio como o mais importante; o utilizador deverá ser devidamente treinado de modo a sentir-se confortável na operação do sistema e a ter capacidade de descobrir erros e/ou omissões nos requisitos; deverá ser elaborado um plano de avaliação com base nos objectivos.

2.4.3. *Reverse Engineering*

Thomas (1994) define *reverse engineering* como a actividade de progressão do código (a última fase do desenvolvimento de *software*), para os documentos de suporte ao projecto, o ponto de partida do processo de desenvolvimento. Esta acti-

vidade requer uma pesquisa continuada do código-fonte e gera produtos como o código reestruturado, dicionários de dados e diagramas de fluxo de dados.

Esta actividade justifica-se, pela existência de programas de milhões de linhas de código que estão mal documentados, através de documentação inexistente, escassa, desactualizada ou desadequada e cujos responsáveis pelo desenvolvimento já saíram da empresa. Justifica-se ainda, nas situações em que o desenvolvimento do sistema existente foi feito ao acaso, ou seguindo um processo demasiado empírico conducente a uma grande obscuridade, relativamente ao pensamento que suportou o desenvolvimento do código. Estes sistemas não podem simplesmente deitar-se fora, pois representam um grande investimento. Por outro lado, é consideravelmente perigoso avançar com um novo desenvolvimento, antes que o sistema existente esteja suficientemente documentado ou tenha mesmo sido reestruturado.

Nestes casos, o processo de reestruturação também se pode desenvolver no âmbito da substituição do «código *spaghetti*» por um «código de limpeza», com o mesmo significado.

2.4.4. Ferramentas de apoio ao desenvolvimento de software

Um problema grave com que se tem que lidar no processo de desenvolvimento de *software* é a complexidade, estreitamente relacionada com a dimensão da tarefa. Há que tentar ultrapassar o problema gerado por metodologias que, como alguém referiu, estão orientadas para o *paperwork*: são vinte volumes que *sintetizam* a análise dos requisitos que, por sua vez, geram cinquenta volumes que resumem o projecto do sistema que por sua vez geram milhões de linhas de código. Segundo Thomas (1994), a esta documentação acresce ainda todo o material relacionado com o método de produção do *software*, isto é: informação sobre os testes executados e informação sobre a configuração acerca das versões dos módulos que integram as versões do sistema.

2.4.4.1. Integrated Project Support Environments (IPSE)

O reconhecimento dos problemas anteriormente referidos levou à aceleração do desenvolvimento das ferramentas CASE, isto é, do *Computer Aided Software Engineering*, as quais apoiam as diversas partes do processo de desenvolvimento de *software*. Normalmente, estas ferramentas facultam um apoio parcial ao desenvolvimento, preocupando-se cada qual com aspectos limitados dos estágios de desenvolvimento. Isso levou a que se pensasse na criação de um ambiente de desenvolvimento que promovesse a integração das ferramentas isoladas, facul-

tando um apoio completo ao ciclo de desenvolvimento do *software*. Esse tipo de ambiente é genericamente designado por *Integrated Project Support Environment* (IPSE) e inclui o apoio às actividades de controlo e gestão necessárias a um grande projecto.

O IPSE, *de per se*, não pode garantir a excelência do produto, por exemplo, numa conjuntura em que nem todos estejam motivados para a qualidade do *software*. O que garante é a uniformidade das práticas e a monitoragem próxima do progresso, através da estruturação em torno de uma base de dados mais flexível que as tradicionais que possa conter documentos em texto, diagramas, *layouts* de ecrãs, fontes de código e programas compilados. Deve ainda poder representar as relações entre diferentes documentos como, por exemplo, a ligação entre um programa fonte, a versão compilada e os respectivos dados de teste.

As dificuldades associadas à utilização dos IPSE derivam da sua grande complexidade. Assim sendo, são de esperar necessidades consideráveis de treino dos recursos humanos, de modo a otimizar a sua exploração. Embora o retorno do investimento seja favorável num grande projecto, é também aí que se deverão tomar maiores cautelas face a uma tecnologia nova, pouco experimentada e que envolve um risco considerável. Por outro lado, os custos de conversão de projectos já existentes são elevados e o conservadorismo e inércia das pessoas são problemas a considerar. Finalmente, a falta de standardização destas ferramentas globais implica o aumento do risco, dado que se criam elos e assumem compromissos difíceis, senão impossíveis, de alterar até ao fim do ciclo de vida do projecto.

2.4.4.2. Ferramentas CASE

De acordo com Thomas (1994), as ferramentas de apoio à programação foram as primeiras a surgir. Entre estas contam-se os *verificadores da sintaxe* que garantem a correcção desta para as várias linguagens de programação, os *programas de edição* que editam o código em formato normalizado, os *compiladores com facilidades de debugging* que detectam erros, emitindo mensagens de aviso ou mesmo corrigindo-os, os *analísadores de fluxo estático* que permitem a análise das estruturas de controlo e dos fluxos dos programas, os *compiladores por excepção* que, através da análise das datas, somente compilam os módulos alterados de um programa, os *geradores de dados de teste* que assumem a fastidiosa tarefa de gerar os dados necessários ao teste de todos os fluxos do programa e as *linguagens de quarta geração* que geram automaticamente os troços de código mais frequentes e típicos como, por exemplo, os programas de geração automática de ecrãs, de listagens e de relatórios normalizados.

Quando se recua no ciclo de desenvolvimento, as notações pioram na sua definição, sendo menos precisas na semântica e na sintaxe, o que dificulta o auxílio

por computador. Existem, contudo, *packages* para apoio às notações gráficas que, para além da possibilidade de desenho, contêm instrumentos de verificação do significado dos diagramas e da correcta utilização da notação.

As dificuldades, inerentes à utilização de diferentes ferramentas que apenas facultam um apoio parcial, traduzem-se na impossibilidade de verificação da consistência entre os vários estágios ou documentos produzidos. Por outro lado, é fundamental que o foco do esforço seja na tarefa a desenvolver e não na ferramenta, pois uma má concepção não melhora com a automatização, originando sempre um mau *software*.

As ferramentas de apoio à análise dos requisitos constam de instrumentos que garantem o cruzamento da informação de diversos documentos de texto e que mantêm listas de sinónimos e das relações estabelecidas.

Finalmente, é importante que se refiram as ferramentas de gestão de projectos existentes que permitem, quer um melhor controle do desenrolar das actividades e do progresso do projecto, quer um melhoramento da monitoria e controlo dos custos.

3. CONCLUSÕES

Este artigo pretendeu dar a panorâmica de uma metodologia de desenvolvimento dos sistemas de informação de suporte informático.

Os principais destinatários são os gestores que, de uma forma ou de outra, venham a estar envolvidos em decisões conotadas com o tema. Por tal, optou-se por um tratamento dirigido a não especialistas, em que se omite o detalhe considerado irrelevante para a análise. Entre os aspectos ignorados incluem-se, por exemplo, os pormenores das técnicas usadas no âmbito da metodologia exposta que, no entanto, os leitores mais interessados poderão facilmente encontrar nas obras citadas na bibliografia, ou em outras similares.

Embora se tenha procurado uma abordagem simples, a complexidade e a seriedade do tema poderão, eventualmente, dificultar o primeiro contacto de alguns dos leitores, pelo que se sugere, aos mais interessados, um tipo de leitura cuidado em que, numa primeira passagem, não haja uma preocupação excessiva em tentar compreender tudo. Naturalmente, a experiência do leitor na área ajudará a entender e a visualizar melhor a mensagem.

Ao pretendermos divulgar os fundamentos dos modernos procedimentos de desenvolvimento de sistemas, procurou-se seguir uma metodologia que, de uma forma pouco sofisticada e compreensível para os leigos, abordasse as principais fases do ciclo de desenvolvimento dos sistemas de informação que, invariavelmente, constituem o suporte omnipresente deste tipo de metodologias. De facto, é opinião do autor que, sem este conhecimento básico, dificilmente se terá um

entendimento satisfatório de metodologias mais complexas, por exemplo, aquelas que se socorrem do auxílio do computador para prosseguir os seus objectivos, como é o caso da *Information Engineering*, entre outras.

A metodologia escolhida para suporte da abordagem foi a *Software Engineering*, devido aos aspectos que se salientam de seguida:

- faculta uma boa perspectiva do ciclo de desenvolvimento dos sistemas de informação, com base no «modelo da queda de água»;
- promove um enquadramento simples das principais técnicas utilizadas, mas de fácil e harmónico entendimento;
- possibilita a utilização manual, embora, neste caso, a orientação para o *paperwork* possa representar um acréscimo do esforço de desenvolvimento;
- apresenta uma formalização menos pesada que, por exemplo, a *Structured Systems Analysis and Design Methodology* (SSADM), o que é uma vantagem para uma primeira abordagem;
- exhibe uma simplicidade que permite uma utilização eficiente no desenvolvimento de aplicações pouco sofisticadas, como é o caso dos *transaction-processing systems*, sem contudo perder a aplicabilidade noutros *decision-support systems* mais complexos (ver §2.2.);
- apresenta a vantagem de se restringir à engenharia de *software*, pois isso delimita bem o âmbito de actuação, promovendo uma abordagem focada; isto não significa que se defenda a omissão dos aspectos referentes ao enquadramento da estratégia dos sistemas de informação, no âmbito do contexto estratégico do negócio, tal como defendem metodologias do tipo da *Information Engineering*, ou do tipo da *IBM's Business Systems Planning*; por outro lado, também não se deverão ignorar os aspectos subjacentes à implantação física dos sistemas de informação, nomeadamente o seu planeamento faseado, a recolha e o tratamento de dados e a formação e o treino dos recursos humanos.

Ao longo do artigo, fez-se a apologia da utilização de metodologias em detrimento de outras abordagens, eventualmente mais económicas e, porventura, mais simples. Há, contudo, alguns aspectos que importa salvaguardar. Não nos devemos esquecer que metodologias e técnicas são meros instrumentos para ajudarem à prossecução dos objectivos organizacionais de um modo mais eficiente. Como tal, a sua aplicação deverá evitar a polarização da atenção, em detrimento daquilo que, de facto, é relevante. Não deverão, pois, ser subvertidos e invertidos os verdadeiros interesses. Em segundo lugar, também, por vezes, os interesses económicos dos consultores cultivam a crença e o mito de que o seguimento disciplinado das diversas abordagens metodológicas conduz inevitavelmente ao sucesso. Na realidade, os exemplos de excelência nem sempre confirmam este aspecto como a única chave do sucesso. Talvez fosse mais sensato e realista considerá-lo

como condição necessária, mas não suficiente. Em terceiro lugar, salienta-se a dificuldade em contratar consultores verdadeiramente independentes, quer pela pressão de interesses económicos, quer pelo condicionamento proporcionado pela sua experiência, formação e treino. Assim sendo, há que ter o cuidado de garantir que as metodologias advogadas e propostas se adequam, realmente, aos problemas em causa, ou de verificar se surgem por outras razões.

Finalmente, uma última palavra para salientar a relevância do papel dos sistemas de informação de apoio à tomada de decisão, bem concebidos, na diminuição da incerteza e, portanto, no fornecimento de informação que permita melhorar o conhecimento, em cada instante, das alternativas de escolha em jogo e das suas potenciais ocorrências. De facto, este aspecto poderá constituir uma vantagem competitiva preciosa, na luta que as organizações, cada vez mais, travam pela sobrevivência, pela sua afirmação com sucesso ou pela excelência.

BIBLIOGRAFIA

- ASHWORTH, C., and GOODLAND, M., 1990, *SSADM: A Practical Approach*, McGraw-Hill Ltd., UK.
- AVISON, D. E. and FITZGERALD, G., 1994, *Information Systems Development*, Alfred Waller Ltd.
- BROOKS, F., *The Mythical Man-Month*, 1982, Addison-Wesley, Reading, Massachusetts.
- BUCKINGHAM, R. A., HIRSCHHEIM, R. A., Land, F. F. and Tully, C.J., 1987, *Information Systems Curriculum: a Basis for Course Design*, in Buckingham *et al.*
- DE MARCO, T., 1978, *Structured Analysis and System Specification*, Prentice-Hall, Englewood Cliffs, N.J.
- FEIGENBAUM, E., and MCCORDUCK, P., 1983, *The Fifth Generation*, Addison-Wesley, Reading, Mass.
- HEAL, B., 1993, *Software Engineering: Software Quality*, The Open University, Milton Keynes.
- INCE, D., and HEKMATPOUR, S., 1987, «Software prototyping – progress and prospects», *Information and Software Technology*, **29** (1), pp. 8-14.
- LEHMAN, M. and BELADY, L., 1985, *Program Evolution. Processes of Software Change*, Academic Press, London.
- MARTIN, J., 1967, *Design of Real-Time Computer Systems*, Prentice-Hall, Englewood Cliffs, N.J.
- RICH, E., May 1984, «The Gradual Expansion of Artificial Intelligence», *IEEE Computer*.
- SOMMERVILLE, I., 1994, *Software Engineering*, Addison-Wesley, Reading, Mass.
- THOMAS, P., 1993, *Software Engineering: Design; Object Oriented Software Design*, The Open University, Milton Keynes.
- THOMAS, P., 1994, *Software Engineering: Software Development; Specification*, The Open University, Milton Keynes.
- YOURDON, E., 1972, *Design of On-Line Computer Systems*, Prentice-Hall, Englewood Cliffs, N.J.