Capítulo 10

O desenvolvimento de programas

'First, the fish must be caught,'
That is easy: a baby, I think, could have caught it.
'Next, the fish must be bought,'
That is easy: a penny, I think, would have bought it.
'Now, cook me the fish!'
That is easy, and will not take more than a minute.
'Let it lie in a dish!'
That is easy, because it already is in it.

Lewis Carroll, *Through the Looking Glass*

A finalidade deste capítulo é a apresentação sumária das várias fases por que passa o desenvolvimento de um programa, fornecendo uma visão global da actividade de programação.

A expressão "desenvolvimento de um programa" é frequentemente considerada como sinónimo de programação, ou de codificação, isto é, a escrita de instruções utilizando uma linguagem de programação. Contudo, bastante trabalho preparatório deve anteceder a programação de qualquer solução potencial para o problema que se pretende resolver. Este trabalho preparatório é constituído por fases como a definição exacta do que se pretende fazer, removendo ambiguidades e incertezas que possam estar contidas nos objectivos a atingir, a decisão do processo a utilizar para a solução do problema e o delineamento da solução utilizando uma linguagem adequada. Se este trabalho preparatório for bem feito, a

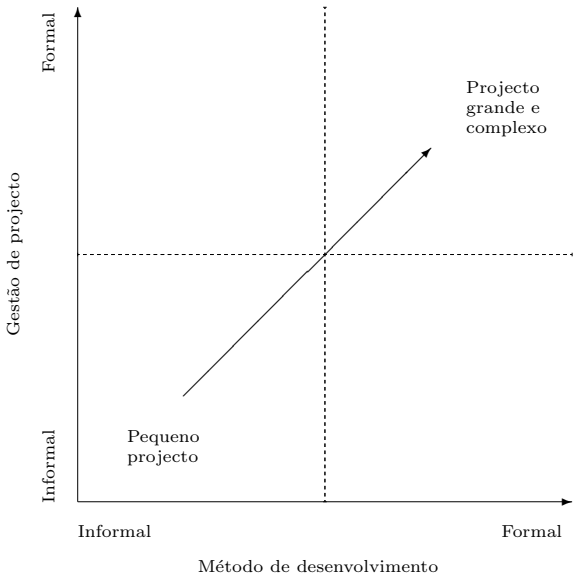

Figura 10.1: O problema da dimensão do programa.

fase de programação, que parece a mais importante para muitas pessoas, torna-se relativamente fácil e pouco criativa. Tal como se despendeu muito trabalho antes de começar a programar, muito trabalho terá de ser feito desde a fase de programação até o programa estar completo. Terão de se detectar e corrigir todos os erros, testar exaustivamente o programa e consolidar a documentação. Mesmo depois de o programa estar completamente terminado, existe trabalho a fazer relativamente à manutenção do programa.

O ponto fundamental nesta discussão é que o desenvolvimento de um programa é uma actividade complexa, constituída por várias fases individualizadas, sendo todas elas importantes, e cada uma delas contribuindo para a solução do problema. É evidente que, quanto mais complexo for o programa, mais complicada é a actividade do seu desenvolvimento.

Desde os anos 60 que a comunidade informática tem dedicado um grande esforço à caracterização e regulamentação da actividade de desenvolvimento de programas complexos. Este trabalho deu origem a uma subdisciplina da informática, a *engenharia da programação*[1] que estuda as metodologias para o desenvolvi-

[1] Do inglês "software engineering".

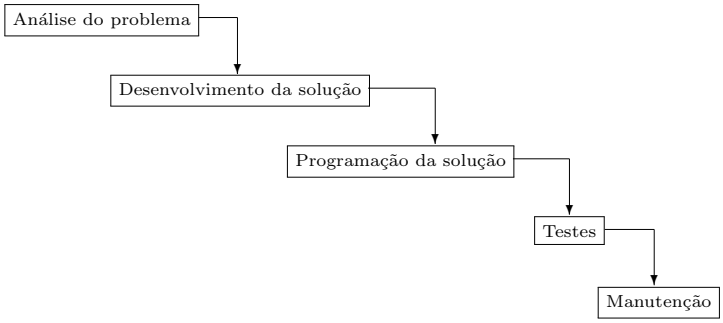

Figura 10.2: Modelo de cascata.

mento de programas. A finalidade da engenharia da programação é a de criar metodologias para desenvolver programas de qualidade a baixo custo. Nesta disciplina faz-se a distinção clara entre o conceito de *programa* (conjunto de instruções escritas numa linguagem de programação) e sistema computacional, entendido como "*software*". A sociedade americana IEEE, "Institute of Electrical and Electronics Engineers", definiu o termo *sistema computacional* como sendo "uma colecção de programas, funções, regras, documentação e dados associados" para transmitir de modo inequívoco que a actividade de programação não se limita à produção de código para ser executado por um computador, mas que inclui toda a documentação e dados associados a esse código.

Existem diferentes metodologias para diferentes tipos de problemas a resolver e diferentes dimensões do programa final. Um dos problemas associados ao desenvolvimento de programas é a dimensão do produto final. Os métodos utilizados para o desenvolvimento de pequenos programas não podem ser adaptados directamente a programas grandes e complexos (Figura 10.1). Quanto maior for a complexidade do programa e a dimensão da equipa associada ao seu desenvolvimento, maior é o nível de formalização necessário para os métodos de desenvolvimento e para a gestão do projecto associado ao seu desenvolvimento.

Uma das metodologias mais usadas no desenvolvimento de programas é baseada no modelo da cascata[2]. De acordo com o *modelo da cascata*, o desenvolvimento de um programa passa por cinco fases distintas que, embora sejam executadas sequencialmente, estão intimamente interligadas (Figura 10.2): a análise do pro-

[2]Do inglês "waterfall method" (ver [Boehm, 1981]).

blema, o desenvolvimento da solução, a programação da solução, os testes e a manutenção. Em todas estas fases é produzida documentação que descreve as decisões tomadas e que serve de apoio ao desenvolvimento das fases subsequentes.

Durante o desenvolvimento de um programa é bom ter sempre em mente a chamada *Lei de Murphy*:

1. Tudo é mais difícil do que parece.
2. Tudo demora mais tempo do que pensamos.
3. Se algo puder correr mal, irá correr mal, no pior dos momentos possíveis.

10.1 A análise do problema

Um programa é desenvolvido para satisfazer uma necessidade reconhecida por um utilizador ou por um conjunto de utilizadores. Neste capítulo, o utilizador (ou utilizadores) é designado por "cliente". As necessidades do cliente são, de um modo geral, apresentadas com lacunas, imperfeições, ambiguidades e até, por vezes, contradições.

Durante a fase da análise do problema, o programador (ou, de um modo mais preciso, o analista – a pessoa que analisa o problema) estuda o problema, juntamente com o cliente, para determinar exactamente *o que tem de ser feito*, mesmo antes de pensar *como os objectivos vão ser atingidos*. Esta fase parece ser de tal maneira óbvia que muitos programadores a ignoram completamente, não perdendo tempo a tentar compreender o problema que se propõem resolver. Como resultado da não consideração desta fase, começam a desenvolver um programa mal concebido, o qual pode representar uma solução incorrecta do problema.

Esta fase processa-se antes de se começar a pensar na solução do problema, mais concretamente, em como resolver o problema. Pretende-se determinar claramente quais as especificações do problema e saber exactamente quais os objectivos a atingir. Esta fase envolve duas entidades com características diferentes, o programador e o cliente. De um modo geral, o programador não tem conhecimento sobre o domínio de aplicação do cliente e o cliente não domina os

aspectos técnicos associados à programação. Este aspecto cria um problema de comunicação que tem de ser resolvido durante a análise do problema.

O trabalho desenvolvido nesta fase inclui um conjunto de interacções entre o programador e o cliente, na qual, progressivamente, ambas as partes aumentam a sua compreensão sobre o que tem que ser feito. Estas interacções são baseadas num documento evolutivo que vai descrevendo, cada vez de modo mais detalhado, quais as características do trabalho a desenvolver.

O resultado desta fase é a criação de um documento – o documento de *análise dos requisitos* – especificando claramente, do ponto de vista informático (mas de modo que sejam perfeitamente entendidos pelo cliente), o que faz o programa, estudo das alternativas para o desenvolvimento do programa e riscos envolvidos no desenvolvimento. Para além deste documento, são também resultados da fase de análise do problema um planeamento pormenorizado para o desenvolvimento com o faseamento das fases seguintes, os custos estimados, etc.

O documento de *análise de requisitos* (conhecido frequentemente por SRS[3]) embora não especifique *como* é que o programa vai abordar a solução do problema, tem duas finalidades importantes. Por um lado, para o cliente, serve de garantia escrita do que vai ser feito. Por outro lado, para o programador, serve como definição dos objectivos a atingir.

10.2 O desenvolvimento da solução

Uma vez sabido *o que* deve ser feito, durante o desenvolvimento da solução determina-se *como* deve ser feito. Esta é uma fase predominantemente criativa, em que se desenvolve um algoritmo que constitui a solução do problema a resolver.

O desenvolvimento deste algoritmo deve ser feito sem ligação a uma linguagem de programação particular, pensando apenas em termos das operações e dos tipos de dados que vão ser necessários. Normalmente, os algoritmos são desenvolvidos utilizando linguagens semelhantes às linguagens de programação nas quais se admitem certas descrições em língua natural. Pretende-se, deste modo, descrever rigorosamente como vai ser resolvido o problema sem se entrar, no entanto, nos pormenores inerentes a uma linguagem de programação.

[3]Do inglês, "System Requirements Specifications".

A ideia chave a utilizar durante esta fase é a *abstracção*. Em qualquer instante deve separar-se o problema que está a ser abordado dos pormenores irrelevantes para a sua solução.

As metodologias a seguir durante esta fase são o *desenvolvimento do topo para a base*[4] e a *refinação por passos*[5], as quais têm como finalidade a diminuição da complexidade do problema a resolver, originando também algoritmos mais legíveis e fáceis de compreender. Segundo estas metodologias, o primeiro passo para a solução de um problema consiste em identificar os principais subproblemas que constituem o problema a resolver, e em determinar qual a relação entre esses subproblemas. Depois de concluída esta primeira fase, desenvolve-se uma primeira aproximação do algoritmo, aproximação essa que é definida em termos dos subproblemas identificados e das relações entre eles. Depois deve repetir-se sucessivamente este processo para cada um dos subproblemas. Quando se encontrar um subproblema cuja solução é trivial, deve-se então escrever o algoritmo para esse subproblema.

Esta metodologia para o desenvolvimento de um algoritmo tem duas vantagens: o controle da complexidade e a modularidade da solução. Quanto ao *controle da complexidade*, devemos notar que em cada instante durante o desenvolvimento do algoritmo estamos a tentar resolver um subproblema único, sem nos preocuparmos com os pormenores da solução, mas apenas com os seus aspectos fundamentais. Quanto à *modularidade da solução*, o algoritmo resultante será constituído por módulos, cada módulo correspondente a um subproblema cuja função é perfeitamente definida.

Seguindo esta metodologia, obtém-se um algoritmo que é fácil de compreender, de modificar e de corrigir. O algoritmo é fácil de compreender, pois é expresso em termos das divisões naturais do problema a resolver. Se pretendermos alterar qualquer aspecto do algoritmo, podemos determinar facilmente qual o módulo do algoritmo (ou seja, o subproblema) que é afectado pela alteração, e só teremos de considerar esse módulo durante o processo de modificação. Os algoritmos tornam-se assim mais fáceis de modificar. Analogamente, se detectarmos um erro na concepção do algoritmo, apenas teremos de considerar o subproblema em que o erro foi detectado.

O resultado desta fase é um documento, o *documento de concepção* (conhecido

[4]Do inglês, "top down design".
[5]Do inglês, "stepwise refinement".

frequentemente por SDD[6], em que se descreve pormenorizadamente a solução do problema, as decisões que foram tomadas para o seu desenvolvimento e as decisões que têm de ser adiadas para a fase da programação da solução. O documento de concepção está normalmente organizado em dois subdocumentos, a concepção global e a concepção pormenorizada. O documento de *concepção global* identifica os módulos principais do sistema, as suas especificações de alto nível, o modo de interacção entre os módulos, os dados que recebem e os resultados que produzem. O documento de *concepção pormenorizada* especifica o algoritmo utilizado por cada um dos módulos e os tipos de dados que estes manipulam.

A fase de desenvolvimento da solução termina com uma verificação formal dos documentos produzidos.

10.3 A programação da solução

The sooner you start coding your program the longer it is going to take.
[Ledgard, 1975]

Antes de iniciarmos esta secção, será importante ponderarmos a citação de Henri Ledgard: "*Quanto mais cedo começares a escrever o teu programa mais tempo demorarás*". Com esta frase, Ledgard pretende dizer que o desenvolvimento de um programa sem um período prévio de meditação e planeamento, em relação ao que deve ser feito e como fazê-lo, leva a situações caóticas, que para serem corrigidas requerem mais tempo do que o tempo despendido num planeamento cuidado do algoritmo.

Só depois de termos definido claramente o problema a resolver e de termos desenvolvido cuidadosamente um algoritmo para a sua solução, poderemos iniciar a fase de programação, ou seja, a escrita do algoritmo desenvolvido, recorrendo a uma linguagem de programação.

O primeiro problema a resolver, nesta fase, será a escolha da linguagem de programação a utilizar. A escolha da linguagem de programação é ditada por duas considerações fundamentais:

[6] Do inglês "Software Design Description".

1. *As linguagens existentes (ou potencialmente existentes) no computador que vai ser utilizado.* De facto, esta é uma limitação essencial. Apenas poderemos utilizar as linguagens que se encontram à nossa disposição.

2. *A natureza do problema a resolver.* Algumas linguagens são mais adequadas à resolução de problemas envolvendo fundamentalmente cálculos numéricos, ao passo que outras linguagens são mais adequadas à resolução de problemas envolvendo manipulações simbólicas. Assim, tendo a possibilidade de escolha entre várias linguagens, não fará sentido, por exemplo, a utilização de uma linguagem de carácter numérico para a solução de um problema de carácter simbólico.

Uma vez decidida qual a linguagem de programação a utilizar, e tendo já uma descrição do algoritmo, a geração das instruções do programa é relativamente fácil. O programador terá de decidir como representar os tipos de dados necessários e escrever as respectivas operações. Em seguida, traduzirá as instruções do seu algoritmo para instruções escritas na linguagem de programação a utilizar. O objectivo desta fase é a concretização dos documentos de concepção.

O resultado desta fase é um programa escrito na linguagem escolhida, com comentários que descrevem o funcionamento das funções e os tipos de dados utilizados, bem como um documento que complementa a descrição do algoritmo produzido na fase anterior e que descreve as decisões tomadas quanto à representação dos tipos de dados.

Juntamente com estes documentos são apresentados os resultados dos testes que foram efectuados pelo programador para cada um dos módulos que compõem o sistema (os chamados *testes de módulo* ou *testes unitários*). Paralelamente, nesta fase desenvolve-se a documentação de utilização do programa.

10.3.1 A depuração

> *Anyone who believes his or her program will run correctly the first time is either a fool, an optimist, or a novice programmer.*
> [Schneider et al., 1978]

Na fase de depuração (do verbo depurar, tornar puro, em inglês, conhecida

por "*debugging*"[7]) o programador detecta, localiza e corrige os erros existentes no programa desenvolvido. Estes erros fazem com que o programa produza resultados incorrectos ou não produza quaisquer resultados. A fase de depuração pode ser a fase mais demorada no desenvolvimento de um programa.

Existem muitas razões para justificar a grande quantidade de tempo e esforço despendidos normalmente na fase de depuração, mas duas são de importância primordial. Em primeiro lugar, a facilidade de detecção de erros num programa está directamente relacionada com a clareza da estrutura do programa. A utilização de abstracção diminui a complexidade de um programa, facilitando a sua depuração. Em segundo lugar, as técnicas de depuração não são normalmente ensinadas do mesmo modo sistemático que as técnicas de desenvolvimento de programas. A fase de depuração é, em grande parte dos casos, seguida sem método, fazendo tentativas cegas, tentando alguma coisa, qualquer coisa, pois não se sabe como deve ser abordada.

Os erros de um programa podem ser de dois tipos distintos, erros de natureza sintáctica e erros de natureza semântica.

A depuração sintáctica

Os erros de *natureza sintáctica* são os erros mais comuns em programação, e são os mais fáceis de localizar e de corrigir. Um erro sintáctico resulta da não conformidade de um constituinte do programa com as regras sintácticas da linguagem de programação. Os erros sintácticos podem ser causados por erros de ortografia ao escrevermos o programa ou por um lapso na representação da estrutura de uma instrução.

Os erros de natureza sintáctica são detectados pelo processador da linguagem, o qual produz mensagens de erro, indicando qual a instrução mais provável em que o erro se encontra e, normalmente, qual o tipo de erro verificado. A tendência actual em linguagens de programação é produzir mensagens de erro que auxiliem, tanto quanto possível, o programador.

A correcção dos erros sintácticos normalmente não origina grandes modificações no programa.

A fase de depuração sintáctica termina quando o processador da linguagem não

[7] Ver a nota de rodapé na página 23.

encontra quaisquer erros de natureza sintáctica no programa. O programador inexperiente tende a ficar eufórico quando isto acontece, não tendo a noção de que a verdadeira fase de depuração vai então começar.

A depuração semântica

Os *erros semânticos* resultam do facto de o programa, sintacticamente correcto, ter um significado para o computador que é diferente do significado que o programador desejava que ele tivesse. Os erros de natureza semântica podem causar a interrupção da execução do programa, ciclos infinitos de execução, a produção de resultados errados, etc.

Quando, durante a fase de depuração semântica, se descobre a existência de um erro cuja localização ou origem não é facilmente detectável, o programador deverá recorrer metodicamente às seguintes técnicas de depuração (ou a uma combinação delas):

1. Utilização de *programas destinados à depuração*. Certos processadores de linguagens fornecem programas especiais que permitem fazer o rastreio[8] automático do programa, inspeccionar o estado do programa quando o erro se verificou, alterar valores de variáveis, modificar instruções, etc.

2. Utilização da técnica da *depuração da base para o topo*[9]. Utilizando esta técnica, testam-se, em primeiro lugar, os módulos (por módulo entenda-se uma função correspondente a um subproblema) que estão ao nível mais baixo, isto é, que não são decomponíveis em subproblemas, e só quando um nível está completamente testado se passa ao nível imediatamente acima. Assim, quando se aborda a depuração de um nível tem-se a garantia de que não há erros em nenhum dos níveis que ele utiliza e, portanto, que os erros que surgirem dependem *apenas* das instruções nesse nível.

Depois de detectado um erro de natureza semântica, terá de se proceder à sua correcção. A correcção de um erro de natureza semântica poderá variar desde casos extremamente simples a casos que podem levar a uma revisão completa

[8] Do dicionário da Porto Editora: rastear *v.t.* seguir o rasto de; rastreio *s.m.* acto de rastear; rasto *s.m.* vestígio que alguém, algum animal ou alguma coisa deixou no solo ou no ar, quando passou.

[9] Do inglês, "bottom-up debugging".

da fase de desenvolvimento da solução e, consequentemente, à criação de um novo programa.

10.3.2 A finalização da documentação

O desenvolvimento da documentação do programa deve começar simultaneamente com a formulação do problema (fase 1) e continuar à medida que se desenvolve a solução (fase 2) e se escreve o programa (fase 3). Nesta secção, vamos descrever o conteúdo da documentação que deve estar associada a um programa no instante em que ele é entregue à equipa de testes (ver Secção 10.4). A documentação de um programa é de dois tipos: a documentação destinada aos utilizadores do programa, chamada documentação de utilização, e a documentação destinada às pessoas que irão fazer a manutenção do programa, a documentação técnica.

A documentação de utilização

A *documentação de utilização* tem a finalidade de fornecer ao utilizador a informação necessária para a correcta utilização do programa. Esta documentação inclui normalmente o seguinte:

1. *Uma descrição do que o programa faz.* Nesta descrição deve estar incluída a área geral de aplicação do programa e uma descrição precisa do seu comportamento. Esta descrição deve ser bastante semelhante à descrição desenvolvida durante a fase de análise do problema.

2. *Uma descrição do processo de utilização do programa.* Deve ser explicado claramente ao utilizador do programa o que ele deve fazer, de modo a poder utilizar o programa.

3. *Uma descrição da informação necessária ao bom funcionamento do programa.* Uma vez o programa em execução, vai ser necessário fornecer-lhe certa informação para ser manipulada. A forma dessa informação é descrita nesta parte da documentação. Esta descrição pode incluir a forma em que os ficheiros com dados devem ser fornecidos ao computador (se for caso disso), o tipo de comandos que o programa espera receber durante o seu funcionamento, etc.

4. *Uma descrição da informação produzida pelo programa*, incluindo por vezes a explicação de mensagens de erro.

5. *Uma descrição, em termos não técnicos, das limitações do programa.*

A documentação técnica

A *documentação técnica* fornece ao programador que irá modificar o programa a informação necessária para a compreensão do programa. A documentação técnica é constituída por duas partes, a documentação externa e a documentação interna.

A parte da documentação técnica que constitui a *documentação externa* descreve o algoritmo desenvolvido na fase 2 (desenvolvimento da solução), a estrutura do programa, as principais funções que o constituem e a interligação entre elas. Deve ainda descrever os tipos de dados utilizados no algoritmo e a justificação para a escolha de tais tipos.

A *documentação interna* é constituída pelos comentários do programa. Os *comentários* são linhas ou anotações que se inserem num programa e que descrevem, em língua natural, o significado de cada uma das partes do programa. Cada linguagem de programação tem a sua notação própria para a criação de comentários; por exemplo, em Python um comentário é tudo o que aparece numa linha após o carácter `#`. Para exemplificar a utilização de comentários em Python, apresenta-se de seguida a função `factorial`, juntamente com um comentário:

```
# Calcula o factorial de um número.
# Não testa se o número é negativo

def factorial(n):
    fact = 1
    for i in range(n, 0, -1):
        fact = fact * i
    return fact
```

Os comentários podem auxiliar tremendamente a compreensão de um programa, e por isso a sua colocação deve ser criteriosamente estudada. Os comentários

devem identificar secções do programa e devem explicar claramente o objectivo dessas secções e o funcionamento do algoritmo respectivo. É importante não sobrecarregar o programa com comentários, pois isso dificulta a sua leitura. Certos programadores inexperientes tendem por vezes a colocar um comentário antes de cada instrução, explicando o que é que ela faz. Os comentários devem ser escritos no programa, à medida que o programa vai sendo desenvolvido, e devem reflectir os pensamentos do programador ao longo do desenvolvimento do programa. Comentários escritos depois de o programa terminado tendem a ser superficiais e inadequados.

Uma boa documentação é essencial para a utilização e a manutenção de um programa. Sem a documentação para o utilizador, um programa, por excepcional que seja, não tem utilidade, pois ninguém o sabe usar. Por outro lado, uma boa documentação técnica é fundamental para a manutenção de um programa. Podemos decidir modificar as suas características ou desejar corrigir um erro que é descoberto muito depois do desenvolvimento do programa ter terminado. Para grandes programas, estas modificações são virtualmente impossíveis sem uma boa documentação técnica.

Finalmente, uma boa documentação pode ter fins didácticos. Ao tentarmos desenvolver um programa para uma dada aplicação, podemos aprender bastante ao estudarmos a documentação de um programa semelhante.

10.4 A fase de testes

Depois de o processo de depuração semântica aparentemente terminado, isto é, depois de o programa ser executado e produzir resultados correctos, poderá ser posto em causa se o programa resolve o problema para que foi proposto para todos os valores possíveis dos dados. Para garantir a resposta afirmativa a esta questão, o programa é entregue a uma equipa de testes, a qual deverá voltar a verificar os testes de cada um dos módulos executados na fase anterior e, simultaneamente, verificar se todos os módulos em conjunto correspondem à solução acordada com o cliente. A equipa de testes deverá criar uma série de casos de teste para o sistema global (tal como foi descrito no documento de concepção global) e testar o bom funcionamento do programa, para todos estes casos.

Os *casos de teste* deverão ser escolhidos criteriosamente, de modo a testarem todos os caminhos, ou rastos, possíveis através do algoritmo. Por exemplo, suponhamos que a seguinte função recebe três números correspondentes aos comprimentos dos lados de um triângulo e decide se o triângulo é equilátero, isósceles ou escaleno:

```
def classifica(l1, l2, l3):
    if l1 == l2 == l3:
        return 'Equilátero'
    elif (l1 == l2) or (l1 == l3) or (l2 == l3):
        return 'Isósceles'
    else:
        return 'Escaleno'
```

Embora esta função esteja conceptualmente correcta, existem muitas situações que esta não verifica, nomeadamente, se `l1`, `l2` e `l3` correspondem aos lados de um triângulo. Para isso terão de verificar as seguintes condições:

1. As variáveis `l1`, `l2` e `l3` têm valores numéricos.
2. Nenhuma das variáveis `l1`, `l2` e `l3` é negativa.
3. Nenhuma das variáveis `l1`, `l2` e `l3` é nula.
4. A soma de quaisquer duas destas variáveis é maior do que a terceira (num triângulo, qualquer lado é menor do que a soma dos outros dois).

Para além destas condições, os casos de teste deverão incluir valores que testam triângulos isósceles, equiláteros e escalenos.

Devemos notar que, para programas complexos, é impossível testar completamente o programa para todas as combinações de dados e portanto, embora estes programas sejam testados de um modo sistemático e criterioso, existe sempre a possibilidade da existência de erros não detectados pelo programador. Como disse Edsger Dijkstra (1930–2002), uma das figuras mais influentes do Século XX no que respeita a programação estruturada, o processo de testar um programa pode ser utilizado para mostrar a presença de erros, mas nunca para mostrar a sua ausência! ("*Program testing can be used to show the presence of bugs, but never to show their absence!*" [Dahl et al., 1972], página 6).

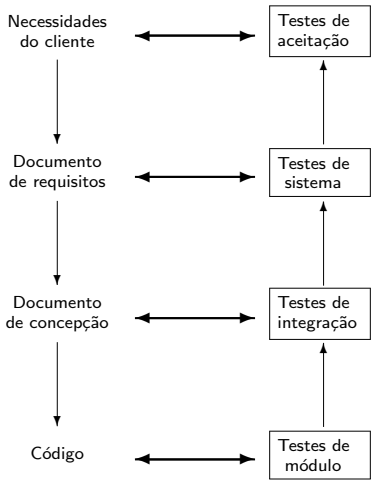

Figura 10.3: Níveis de testes.

Em resumo, os testes de um programa são efectuados a vários níveis (Figura 10.3). Os *testes de módulo* são efectuados pelo programador durante a fase da programação da solução. Os *testes de integração* são efectuados pela equipa de testes, tendo em atenção o documento de concepção. Após a execução, com sucesso, dos testes de integração, a equipa de testes deverá também verificar se o sistema está de acordo com o documento dos requisitos, efectuando *testes de sistema*. Finalmente, após a entrega, o cliente efectua *testes de aceitação* para verificar se o sistema está de acordo com as suas necessidades.

Existem métodos para demonstrar formalmente a correcção semântica de um programa, discutidos, por exemplo, em [Dijkstra, 1976], [Hoare, 1972] e [Wirth, 1973], mas a sua aplicabilidade ainda se limita a programas simples e pequenos.

10.5 A manutenção

Esta fase decorre depois de o programa ter sido considerado terminado, e tem duas facetas distintas. Por um lado, consiste na verificação constante da possibilidade de alterações nas especificações do problema, e, no caso de alteração de especificações, na alteração correspondente do programa. Por outro lado, consiste na correcção dos eventuais erros descobertos durante o funcionamento

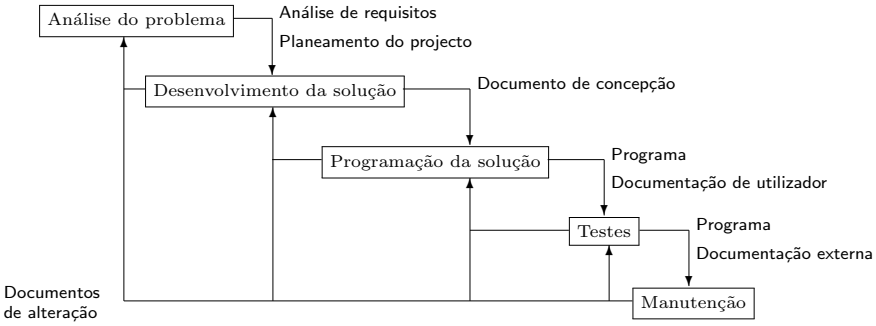

Figura 10.4: Modelo de cascata e documentos associados.

do programa. Em qualquer dos casos, uma alteração no programa obriga a uma correspondente alteração na documentação. Um programa com erros de documentação ou com a documentação desactualizada pode ser pior do que um programa sem documentação, porque encoraja o programador a seguir falsas pistas.

Segundo Frederick P. Brooks, Jr.[10], o custo de manutenção de um programa é superior a 40% do custo total do seu desenvolvimento. Este facto apela a um desenvolvimento cuidado do algoritmo e a uma boa documentação, aspectos que podem diminuir consideravelmente o custo de manutenção.

10.6 Notas finais

Neste capítulo apresentámos de um modo muito sumário as fases por que passa o desenvolvimento de um programa. Seguimos o modelo da cascata, segundo o qual o desenvolvimento de um programa passa por cinco fases sequenciais. Discutimos que a actividade desenvolvida em cada uma destas fases pode levar à detecção de deficiências em qualquer das fases anteriores, que deve então ser repetida.

Esta constatação leva a uma reformulação do modelo apresentado, a qual se indica na Figura 10.4. Nesta figura mostramos que o trabalho desenvolvido em qualquer fase pode levar a um retrocesso para fases anteriores. Indicamos também os principais documentos produzidos em cada uma das fases. Na fase da manutenção podem verificar-se grandes alterações ao programa, o que pode

[10] Ver [Brooks, 1975].

levar a um retrocesso para qualquer uma das fases de desenvolvimento.

O assunto que abordámos neste capítulo insere-se no campo da Engenharia Informática a que se chama *Engenharia da Programação*. Informação adicional sobre este assunto poderá ser consultada em [Jalote, 1997] ou [Sommerville, 1996]. A Engenharia da Programação é uma das áreas da Engenharia Informática em que existe mais regulamentação sobre as metodologias a utilizar. Em [Moore, 1998] encontra-se uma perspectiva global das normas existentes nesta disciplina.