Especificação Formal de Software

Carlos Figueiredo, Jorge Neves, Luís Magalhães, Vitor Pinto

Licenciatura em Engenharia Informática e Computação

Faculdade de Engenharia da Universidade do Porto

E-mail: {ei99030, ei99040, ei99049, ei99036}@fe.up.pt

Resumo

Este relatório visa reflectir a aplicação de métodos formais na Engenharia de Software, bem como as vantagens e desvantagens do uso deste tipo de métodos. Será ainda efectuada uma breve descrição das principais linguagens necessárias à elaboração de um método formal.

1 Introdução

Com a larga e crescente utilização informática nos dias de hoje, é cada vez maior a dependência dos computadores *(hardware)* e seus aplicativos *(software)*. À medida que essa dependência aumenta, aumentam também as complexidades dos projectos a conceber. Uma questão fundamental no desenvolvimento de um software é garantir que este é realmente uma solução para o problema proposto. É neste ponto que os métodos formais são importantes para a Engenharia de Software.

As disciplinas tradicionais de Engenharia baseiam-se fortemente em modelos matemáticos para atingirem os seus objectivos. Ao contrário das suas congéneres, a Engenharia de Software tem uma abordagem muito menos rigorosa e consequentemente sofreu e continua a sofrer importantes fracassos que têm uma repercussão cada vez mais importante na sociedade. A área de métodos formais tem vindo a contrariar esse quadro.

Os métodos formais foram desenvolvidos para auxiliar todas as etapas da elaboração de um software: especificação formal para a definição dos requisitos, refinamento para a concepção, síntese para a codificação, prototipagem para a validação, prova para a verificação.

A figura 1 mostra a forma como o projecto de especificação formal se enquadra na estrutura tradicional de desenvolvimento de software. Criar uma especificação formal detalhada força uma análise pormenorizada do sistema que revela erros e inconsistências na especificação informal, este permite assim eliminar ambiguidades e aumentar a consistência e o grau de “completicidade” dos requisitos a serem atingidos, argumentos que outros métodos convencionais não conseguem atingir ou pelo menos com grande precisão.

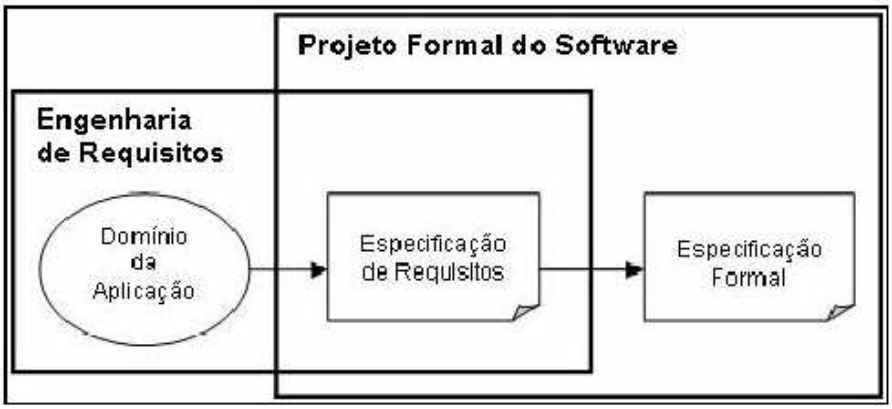

Figura 1 – Especificação de Requisitos e Especificação Formal

Estes argumentos destacaram os métodos formais dos outros e devem-se ao uso da matemática, uma linguagem precisa e coerente, como base na criação de modelos. Essa precisão permite diminuir drasticamente os erros que podem vir a surgir na elaboração de *software's*. No entanto não elimina de todo os erros, pois não é possível criar um sistema perfeito, isento de problemas e falhas. É devido a este factor que existe ainda alguma controvérsia em torno dos métodos formais, no entanto é unânime que o uso deste tipo de métodos nas especificações de requisitos melhora grandemente a qualidade do trabalho a desenvolver e por conseguinte o produto final.

O uso crescente de métodos formais levou a que dessem origem à criação de notações próprias, de modo a serem usadas pelos engenheiros de software com alguma padronização, notações como a Z ou VDM tornaram-se obrigatórias para qualquer engenheiro de software que queira recorrer ao uso de métodos formais, na elaboração dos seus sistemas de requisitos.

2 Importância do uso de Linguagens de Especificação

Actualmente a grande maioria do software possui uma complexidade muito elevada, o que cria a necessidade de definir com rigor as relações entre os vários componentes, bem como o funcionamento interno de cada um dos componentes que constituem o software, para que não existam falhas. Neste contexto, as linguagens de especificação, ao permitirem definir com rigor matemático os problemas para os quais se espera que o software responda, permitem criar um software com maior fiabilidade.

Em seguida irão ser descritas as vantagens, as desvantagens (nesta secção não serão abordadas somente as desvantagens, mas também alguns dos motivos pessoais que levam os responsáveis envolvidos num projecto a não escolher estes métodos) e alguns mitos relativos à especificação formal.

2.1 Vantagens

O desenvolvimento de uma especificação formal permite ao projectista compreender melhor os requisitos do sistema, bem como a forma de os implementar num

software funcional, com um número reduzido de erros e omissões na implementação da solução.

Como se refere a uma linguagem matemática, a especificação formal é precisa e não ambígua.

As especificações formais podem ser analisadas matematicamente de forma a demonstrar a sua consistência e o seu grau de "completicidade". Após a implementação pode ser possível provar que o software foi desenvolvido de acordo com a especificação inicialmente criada.

A especificação formal pode ser utilizada para identificar os casos de teste mais relevantes para o software a desenvolver.

As especificações formais podem ser processadas automaticamente, através de ferramentas de software que permitem o seu desenvolvimento, compreensão e *debug*. Utilizando estas ferramentas é possível simular um protótipo do sistema, embora esta simulação não seja muito eficiente em termos de rapidez.

O custo final do software pode ser reduzido, uma vez que estes métodos permitem corrigir erros numa fase inicial do projecto, o que se torna mais barato que a correcção após a implementação do mesmo.

2.2 Desvantagens

Devido ao conservadorismo existente entre os gestores de *software,* existe alguma relutância em adoptar novas técnicas quando o lucro não é óbvio. É difícil demonstrar que o elevado custo de desenvolvimento da especificação formal, na fase inicial, irá tornar o preço final do projecto mais baixo.

Dado que muitos engenheiros de *software* não têm experiência em matemática discreta e em lógica, têm alguma dificuldade em desenvolver especificações formais.

Os clientes não estão familiarizados com as linguagens de especificação formal, o que os conduz ao não financiamento de projectos especificados com este tipo técnicas.

Os métodos existentes actualmente não permitem a especificação dos componentes interactivos das interfaces do utilizador e são difíceis de utilizar em alguns sistemas paralelos.

A maioria do trabalho desenvolvido na especificação formal é dirigido ao desenvolvimento de línguas, existindo poucas ferramentas que as implementem.

2.3 Mitos dos métodos formais

Aqui serão expostos algumas crenças relativas aos métodos formais que foram exageradas e para as quais iremos apresentar a argumentação que Hall [9] utilizou para desmistificar estas crenças.

- **Os Métodos formais podem garantir que o software seja perfeito**

  Este mito, obviamente não é verdade, pois uma especificação formal é um modelo do mundo real e portanto pode incluir alguns erros ou omissões na representação do mundo.

- **Os Métodos formais só são utilizados para prova do programa**

Os métodos formais também são usados para escrever uma especificação formal, provar as propriedades das especificações e construir um programa manipulando matematicamente as especificações.
Os métodos formais fornecem uma abstracção da especificação. Descrevem o que tem de ser feito e não como o fazer e permite especificar o detalhe apenas ao nível necessário.

- **Devido ao elevado custo, os métodos formais só são úteis para sistemas de segurança críticos**

Geralmente os métodos formais reduzem os custos para todas classes de sistemas e não só para os sistemas de segurança críticos.

- **São necessários matemáticos treinados para a implementação dos métodos formais**

A matemática usada é simples, embora necessite de algum treino, como qualquer outro método. A maior dificuldade reside na obtenção do nível de abstracção ideal para modelar o mundo real.
A realização da prova formal já exige um nível mais elevado de conhecimentos matemáticos.

- **Os Métodos Formais aumentam o custo de desenvolvimento**

Os métodos formais alteram a estrutura do desenvolvimento ao despender-se mais tempo na fase a especificação e menos tempo nas fases seguintes, aumentando, assim o custo da fase inicial, mas diminuindo o custo total do projecto.

- **Os Métodos Formais são inaceitáveis pelos utilizadores**

Os métodos formais ajudam os utilizadores porque a especificação captura o que o utilizador quer antes de ser construído. A especificação formal pode ser compreendida pelo cliente, se for traduzida para linguagem natural e através da criação de protótipos que exemplifiquem a especificação.

- **Os Métodos Formais só são usados em sistemas triviais**

Os métodos formais já foram utilizados para especificação de um *kernel* de um sistema de tempo-real [10] e para a especificação de um osciloscópio [11].

3 Linguagens de Especificação

Esta secção pretende especificar de uma forma geral os fundamentos, vantagens e desvantagens, dos diferentes tipos de linguagens de especificação.

Existem várias linguagens de implementação de métodos formais. A figura 2 apresenta algumas dessas linguagens e implementa uma divisão dessas linguagens [2]. Esta divisão tem em conta as características das linguagens, nomeadamente relacionadas com os fundamentos científicos em que se apoia a especificação. Como característica comum, salienta-se que todas as linguagens são constituídas por: uma notação, que especifica o domínio sintáctico; um universo de objectos, que refere o domínio

semântico; uma regra que indica quais os objectos que satisfazem cada especificação. Nos pontos seguintes, apresentam-se cada um dos diferentes tipos de linguagem.

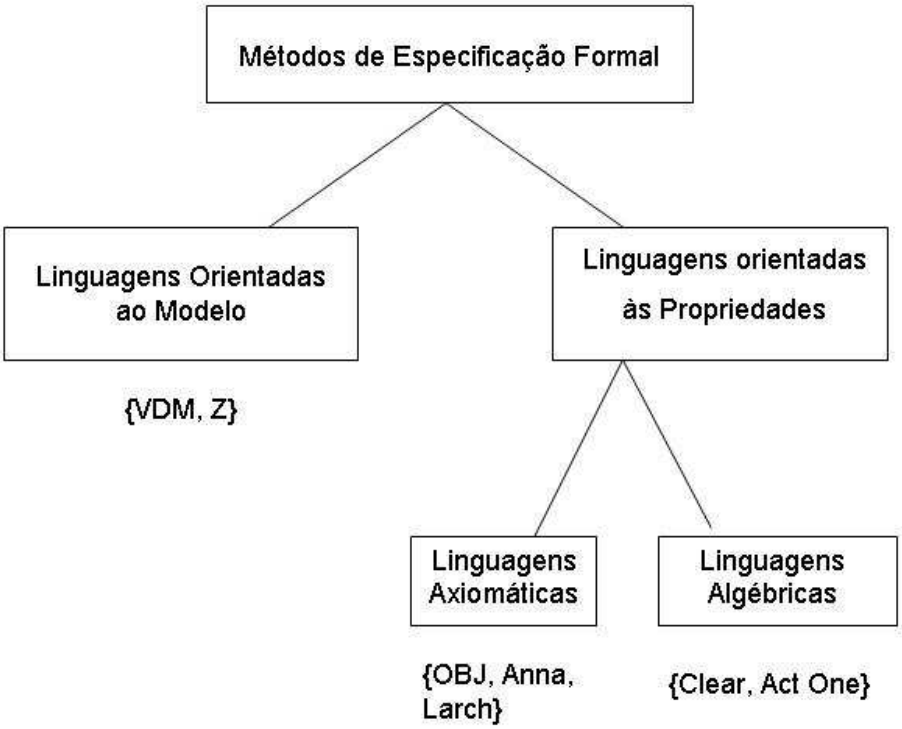

Figura 2 – Diferentes abordagens dos métodos formais

3.1 Linguagem Orientadas ao Modelos

Este tipo de linguagens usam modelos matemáticos construídos usando tipos de dados simples como listas, caracteres, e números naturais, são ainda especificadas as operações que mudam o estado do modelo. Ou seja estas linguagens representam o "mundo" que pretendem especificar como um conjunto de estados.

Existem dois representantes deste tipo de linguagens que têm sido bastante usados e que são bastantes poderosos, o VDM e o Z (pronuncia-se *Zed*). Seguidamente é feita uma apresentação destas linguagens.

A linguagem VDM (*The Vienna Development Method*) foi desenvolvida inicialmente pelo laboratório da IBM em Viena, em meados da década de 1970, com o objectivo de ajudar o desenvolvimento de Sistemas Operativos [1]. Até aos tempos que correm foram feitas varias actualizações à notação e as ferramentas que a implementam, o que torna esta linguagem numa ferramenta poderosa que tem sido aplicada no desenvolvimento de vários sistemas. Actualmente esta linguagem encontra-se estandardizada pelo ISO, com o nome VDM-SL.

Esta linguagem usa um conjunto de regras que permitem refinar os dados e as operações, possibilitando desta forma interligar os requisitos abstractos do sistema com os detalhes de implementação relacionados com o código escrito [3]. Ou seja a especificação contem o tipo de dado usado e as operações que são permitidas. Existem várias extensões ao VDM, entre as quais RSL e VDM++, esta última possibilita a inclusão do conceito de programação orientada a objectos. Estas extensões para além de incluírem várias funcionalidades extra-standard, normalmente têm como alvo, um tipo específico de software.

Já foi citada a forma como era construída uma especificação, de seguida apresenta-se um exemplo. Suponha-se um sistema de reservas de um hotel, semelhante ao usado nas aulas práticas de ES. Este sistema seria especificado por listas que representariam os quartos, os clientes e a ocupação dos quartos. A especificação incluiria também operações para adicionar, remover e alterar os dados constantes das listas.

Existem muitas semelhanças entre a linguagem anterior e a linguagem Z, [4] algumas das diferenças prendem-se com pormenores de implementação. A linguagem Z é estruturada como um conjunto de esquemas. Um esquema é uma representação formal de estados que é usada para estruturar a especificação formal. Num esquema, são introduzidas variáveis e são especificadas as relações entre variáveis.

Os esquemas são vistos como construtores, que descrevem os estados do sistema. Cada estado pode ser constituído por várias operações, estas operações resultam na mudança entre os diferentes estados. Desta forma o sistema que se pretende implementar é visto como uma máquina de estados [5].

3.2 Linguagens Baseadas em Propriedades

Este tipo de linguagens especifica o comportamento do sistema de uma forma indirecta, através da especificação de um conjunto de propriedades que o sistema deve satisfazer. Enquanto as linguagens baseadas em modelos especificam o sistema com base no seu funcionamento (máquina de estados), as linguagens baseadas em propriedades, preocupam-se mais em identificar as propriedades externas do sistema e em identificar o que este deve fazer. Desta forma as linguagens baseadas em propriedades relacionam-se mais com a especificação dos requisitos. Existem duas subcategorias deste tipo de linguagens, as axiomáticas e as algébricas. De seguida descreve-se este tipo de linguagens.

3.2.1 Linguagens Axiomáticas

Este tipo de linguagens usa abstracções do sistema baseadas na lógica. Algumas destas linguagens são, OBJ e Larch.

A linguagem OBJ teve o seu início na década de 70 do século passado, sendo assim pioneira entre as linguagens de especificação. A linguagem OBJ é vista como uma família, da qual fazem parte os seguintes membros, OBJ3, CafeOBJ e o BOBJ. Estas linguagens de especificação usam conceitos de lógica e disponibilizam um poderoso método de especificação. De uma forma geral pode-se caracterizar estas linguagens como

sendo um conjunto de sentenças de um sistema lógico. Do sistema fazem ainda parte operações semânticas deduzidas a partir do sistema lógico. Estas linguagens usam também alguns conceitos de álgebra com o objectivo de definir, algumas características da linguagem como por exemplo tratamento de excepções [7].

A linguagem de especificação Anna (***Ann****otated* ***A****da*) é o resultado de anos de pesquisa em validação de programas. Esta linguagem é uma extensão da linguagem Ada, sendo que já esta é muito poderosa, tendo como principais características, a grande diversidade de tipos de dados suportados e o tratamento de excepções. Esta linguagem baseia-se na especificação por "comentários formais", estes podem ser Anotações, especificações de elementos do programa, ou podem ser Texto Ada, que consiste em texto específico da linguagem, por exemplo, uma excepção conhecida ou uma definição de um *package*. As anotações podem ser por exemplo especificação de objectos.

3.2.2 Linguagens Algébricas

Uma das características menos atractivas dos métodos formais deve-se à relativa complexidade das notações utilizadas. Com o objectivo de simplificar as notações começaram-se a usar linguagens algébricas, usando axiomas na forma de equações mais simples e mais acessíveis a todos. [2]

Um termo correntemente usado em especificações é o termo *sort*. Uma entidade ou um objecto é visto como uma instância de um *sort*. Este conceito está relacionado com a ideia de tipos ou classes, por outras palavras um *sort* é um conjunto de objectos.
A especificação algébrica divide-se em quatro módulos:

- Introdução
- Descrição informal
- Assinatura
- Axiomas

Introdução – Na introdução é definida a entidade a ser usada, juntamente com o *sort* e o nome de outras aplicações *(import)* que irão ser necessárias. Como exemplo se pretendesse-mos definir a "classe" tabela_aviões definiríamos do seguinte modo:

Tabela_avioes (detalhes)
Sort *Tabela_avioes*
Imports *integer, pistas_avioes.*

Descrição informal – Neste módulo introduzem-se comentários textuais que explicam a matemática formal aplicada. Este módulo é extremamente importante pois permite saber o que determinado método faz, facilitando o trabalho ao analista, pois não terá que analisar toda a matemática formal existente, para saber a sua finalidade.

Assinatura – O objectivo da assinatura consiste em definir as características de um objecto, através da descrição das suas propriedades, usando um conjunto de operações. Estas operações dividem-se em *constructor e inspection*. O *constructor* é usado para criar e alterar entidades definidas no *sort* (classe) em questão.
O *inspection* relaciona-se com as operações usadas. Este tem como finalidade avaliar as atribuições resultantes de operações e verificar a validade dessas mesmas operações.

Operações como *create, insert e remove* são claramente instruções de *constructor*, uma vez que estão relacionadas com a criação, eliminação de objectos. Outras como *last e eval* são claramente operações de *inspection*, pois tem que ser verificada a veracidade e validade da operação.

Axiomas – É neste módulo que se utilizam as notações matemáticas, as operações a serem efectuadas e que serão avaliadas pelo *inspector* e as relações entre elas. É aqui que residem as especificações algébricas propriamente ditas. Estas especificações consistem basicamente num conjunto de equações baseadas em expressões matemáticas.

Os axiomas definem o efeito resultante de uma operação aplicada a um objecto, quando este se encontra num determinado estado. No entanto em vez de se usar um modelo explícito para definir um estado (como acontece com o VDM), este é definido em termos das operações resultantes do construtor *(construtor)*. Desde modo, cada axioma relata o efeito de uma operação do inspector *(inspector)*.

A estratégia de desenvolver especificações algébricas, para serem aplicadas em métodos formais, é mais flexível que o uso de linguagem alternativas como o VDM. Ao contrário desta ultima linguagem, baseada em modelos matemáticos, as linguagens algébricas, orientada às propriedades, enquadram-se mais com a especificação dos requisitos. No entanto é provavelmente mais difícil de construir uma especificação algébrica formal para sistemas complexos, sem se ter uma vasta e rica experiência e conhecimento.

4 Conclusões

Embora os métodos formais permitam construir software mais fiável e eficiente e com um custo final geralmente mais reduzido, estes são pouco utilizados, devido à relutância que os gestores de software têm em financiar novos métodos quando estes não reduzem os custos de forma imediata. Os métodos formais também não permitem descrever os componentes interactivos das interfaces com o utilizador, que actualmente constituem uma parte essencial do software.

Referencias

[1] http://www.csr.ncl.ac.uk/vdm/
[2] David Budgen, *Software design,* 1994 Addison-Wesley, p 337-362
[3] http://www.cs.tcd.ie/FME/original/
[4] J. Hayes, C. B. Jones and J. Nicholls.*Understanding the Differences Between VDM and Z.* disponível via web a partir de:
http://citeseer.nj.nec.com/hayes93understanding.html
[5] Roger S.Pressman *Software Engineering – a Practitioner's Approach European Adaptation fifth edition* – capitulo 25
[6]Goguen J. A. (1994) *"Requirements Engineering as the Reconciliation of Technical and Social Issues"*
[7] http://www.cs.ucsd.edu/users/goguen/sys/obj.html
[8] http://www.sds.lcs.mit.edu/spd/larch/
[9] Hall, A. (1990) *Seven myths of formal methods*. *IEEE Software*, 7(5), 11-20 [123,160,161].
[10] Spivey, J. M. (1990) *Specifying a real-time kernel*. *IEEE Software*, 7(5), 21-8 [162,202].
[11] Delisle, N. and Garlan, D. (1990) *A formal specification of an oscilloscope*. *IEEE Software*, 7(5), 29-36 [162].
[12] Sommerville, Ian (1995) *Software Engineering.* 9, 157-170.
[13] Shimizu, Kanna. *A Methodology for Writing and Exploiting Formal Specifications. Stanford University*