Classificação e Selecção de Componentes em Projectos de Software

Pedro Reis dos Santos
Rui Gustavo Crespo
Instituto Superior Técnico

Resumo

O desenvolvimento de software é ainda uma tarefa exigente para os engenheiros de software apesar das ferramentas e ambientes de desenvolvimento existentes. A escolha de componentes para reutilização, quando existem soluções alternativas, é frequentemente intuitiva com base em critérios subjectivos com origem na experiência anterior. Este trabalho propõe a utilização de identificadores, obtidos nos diversos componentes candidatos a reutilização, classificando-os por forma a realizar testes comparativos que permitam uma escolha mais esclarecida. A mesma técnica é utilizada para permitir detectar inconsistências, entre diversas fases do mesmo processo de desenvolvimento, por comparação dos atributos usados na classificação.

1 Introdução

A reutilização de software é quase tão antiga como o próprio software. Os sistemas operativos são um dos primeiros exemplos de reutilização em larga escala. A reutilização de software pode aumentar significativamente a qualidade e a produtividade do desenvolvimento de software. Com o aumento da dimensão dos projectos de software, a reutilização de componentes desempenhará um papel cada vez mais importante no desenvolvimento de software. O desenvolvimento de software com base apenas em componentes próprios (*in-house syndrome*) não é, ao contrário do que é frequente pensar, um dos factores que mais afectam a não reutilização de software[11]. De igual forma, a linguagem de programação escolhida, a experiência do engenheiro de software, a utilização de ferramentas CASE ou problemas legais pouca influência têm na reutilização.

Factores dominantes na reutilização de software incluem a existência de componentes de alta qualidade e de confiança, a existência de um processo de desenvolvimento de software comum e um ensino que promova a reutilização. O ensino da reutilização pode ser visto como o motor de todo o processo, mas não pode só por si produzir resultados significativos, necessitando de suporte apropriado. Esse suporte deve ser construído em torno de um processo de desenvolvimento de software comum por forma a poder guiar um engenheiro de software experiente (treinado). No entanto, a existência de componentes de alta qualidade não depende nem do processo de desenvolvimento de software nem do engenheiro. Contudo, o processo de desenvolvimento de software deve ajudar o engenheiro na tarefa de identificar os componentes de alta qualidade. Desta forma, o maior esforço deve ser colocado em disponibilizar um processo de desenvolvimento de software que permita ao engenheiro treinado aumentar a reutilização e a qualidade da mesma através da correcta identificação de componentes de boa qualidade[9].

A representação e especificação de componentes de software reutilizáveis exige técnicas de classificação que permitam de uma forma automática ou assistida a selecção dos mesmos. A selecção só poderá ser automática se os requisitos corresponderem precisamente às características dos componentes. Caso contrário, ter-se-á de encontrar os componentes que mais se aproximam dos requisitos e alterá-los subsequentemente. Por vezes as características que descrevem os componentes estão incorrectamente identificadas ou incompletas. Neste caso, podem-se retirar conclusões enganadoras, parecendo mais atractivos componentes que são na prática piores soluções.

Existe pois uma necessidade para um motor de busca que permita seleccionar um conjunto de possíveis soluções. Não se exige que a selecção seja óptima, nem mesmo boa. Caso existam alternativas deve ser possível escolher com base em parâmetros que não faziam parte da selecção, sem que para isso seja necessário inspeccionar directamente o componente. O equivalente de um ponto de vista médico traduz-se pela selecção de um conjunto de possíveis doenças, com base num conjunto de sintomas previamente identificados.Talvez a resposta mais provável não seja a mais correcta de entre as seleccionadas. Alguns dos sintomas podem não ter sido ainda identificados ou incorrectamente

identificados, conduzindo a respostas pouco precisas. Neste ponto torna-se necessária a intervenção humana com o objectivo de verificar os sintomas e procurar indícios específicos que permitam concluir com maior rigor qual a doença. Embora alguns dos sintomas possam nunca se verificar, a escolha final do tratamento cabe ao médico. Este deve escolher, de entre as possíveis doenças aquela que mais se aproxima dos sintomas do doente.

O objectivo deste trabalho reside em descrever componentes de software por forma a permitir pesquisas automáticas e buscas manuais esclarecidas, a fim de seleccionar o componente que melhor se adapte às necessidades. A componente humana foi especialmente tida em conta, uma vez que pessoas ou técnicas de trabalhos diferentes podem fazer escolhas diferentes. O simples facto de as escolhas recairem sobre componentes distintos não resulta, necessariamente, de um erro de selecção. Experiências e métodos de trabalho diferentes podem fazer escolhas diferentes sem que isso implique uma deterioração da qualidade do produto final ou dos custos para o desenvolver.

2 Trabalho Relacionado

Os componentes de software podem ser descritos a diversos níveis que podem ir desde os requisitos ao código propriamente dito. A forma mais simples de descrever um componente é textualmente, em português ou inglês, normalmente sob a forma de manual. Esta pode ser uma forma aceitável para componentes de alta qualidade e que exibam uma elevada estabilidade temporal. No entanto, mesmo nestes casos, frequentes erros de interpretação conduzem não só a uma selecção menos boa como a uma utilização incorrecta [7]. Mais frequente é, contudo, a discrepância entre o conteúdo do manual e o que o software realmente faz. Este facto é especialmente preocupante quando surge em produtos ditos de qualidade. A documentação representa frequentemente aquilo que se pretende que o produto realize e não aquilo que ele efectivamente faz. Por outro lado, pouco software se pode considerar completamente estável e não sujeito a alteração, excepto quando já está completamente ultrapassado e consequentemente inútil. Embora existam técnicas que permitam manter a documentação sincronizada com o código, estas exigem vastos recursos humanos o que aumenta o custo e o tempo de desenvolvimento do produto. Por outro lado, a documentação é difícil de analisar com o auxílio de ferramentas, com o objectivo de identificar certos parâmetros chave.

Técnicas de classificação oferecem uma forma simples e útil de catalogar componentes de software. Esquemas de classificação simples recorrem a técnicas enumerativas, tal como o sistema decimal de Dewey, mas são muito vagas para aplicações de selecção concretas pois não permitem representar inúmeros pormenores. No entanto, podem ser úteis como primeira aproximação.

Esquemas de classificação baseados em facetas[15] usam várias facetas ou vistas em que cada uma pode conter diversos termos. Para classificar um item, um conjunto de termos que melhor descreve o componente deve ser escolhido de entre uma lista de possibilidades. Esquemas baseados em facetas oferecem uma forma mais rica de classificação que os esquemas enumerativos, uma vez que um componente não é descrito apenas por um só termo mas por um ou mais termos por faceta. No entanto, a estereotipagem dos termos traduz-se em que componentes semelhantes possam ter exactamente a mesma classificação. Esta característica pode até ser considerada positiva numa primeira aproximação selectiva, mas verifica-se inútil quando se torna necessário comparar e identificar pormenores.

Quaisquer das soluções acima descritas podem ser incluidas no grupo das descrições textuais uma vez que se baseiam na intuição humana. Embora métodos automáticos tenham sido tentados a sua aplicação é muito limitada [24].

Comparação de assinaturas, por outro lado, é uma aproximação mais formal, beneficiando do facto de o software, mesmo que possa ser visto como texto, é de facto bastante estruturado. A ideia baseia-se em descrever os componentes ao nível da interface. Quando efectuado de uma forma simplista, a comparação limita-se a características sintacticas do componente. Um pouco à semelhança dos mecanismos usados pelas linguagens para oferecer compilação separada. Soluções menos restritivas têm sido propostas [25] por forma a eliminar restrições sintacticas tais como a ordem dos parâmetros, composição de tipos agregados ou subtipificação. A comparação de tipos agregados definidos pelo utilizador, hierarquias de tipos, parâmetros opcionais ou parâmetros genéricos torna-se muito complicada sendo difícil chegar a conclusões correctas.

A partição por domínios [24,26] tem sido pouco utilizada e pode ser considerada uma extensão natural da comparação de assinaturas. Esta técnica baseia-se no enriquecimento da descrição com origem na assinatura com informação semântica. O trabalho descrito neste documento pode ser incluido neste grupo.

Finalmente, técnicas como os tuplos característicos ou os elementos repartidos [24] recorrem a exemplos particulares de utilização. Estas soluções têm um curto espectro de utilização uma vez que se torna difícil, se não mesmo impossível, descrever componentes através do seu comportamento em casos particulares. Estas técnicas são apenas utilizáveis em componentes simples ou em funções ou objectos isolados.

3 Descrição de Componentes

Numa primeira aproximação torna-se necessário determinar qual a informação relevante e que deve ser utilizada na descrição do componente. A informação relevante deve depois ser representada de uma forma compacta de tal forma que seja significativamente mais fácil analisá-la que analisar o componente propriamente dito.

A abstracção é uma técnica fundamental para compreender e resolver problemas [21, 13]. Se descrições muito concretas são vastas e difíceis de analisar, abstracções suprimem informação que pode vir a ser importante. Além disso, o realismo e pormenor necessário pode variar de caso para caso, para um mesmo componente. Sob este ponto de vista uma abstracção pode ser considerada boa em determinado contexto e fraca noutro. Este trabalho procura aumentar a flexibilidade da representação através do registo da informação a diferentes níveis de granularidade. Como estes níveis são dependentes do tipo de aplicação o mecanismo de selecção torna-se especialmente importante. A utilização de um conjunto de indentificadores pré-definidos pode ser uma solução viável em situações concretas que têm origem em casos bem estudados. No entanto, descrever uma entidade usando apenas identificadores pré-definidos [8] é muito limitativo.

A aproximação descrita não usa, em princípio, identificadores ou vistas pré-definidos. Na realidade, alguns identificadores, utilizados como pontos de entradas, são necessários. Desta forma, os identificadores utilizados, não necessitam ser aqueles que melhor se aproximam de entre os disponíveis, mas os identificadores efectivamente utilizados. A ausência de estereotipos permite maior realismo e expressividade, facilitando a selecção manual mas dificultando a utilização de métodos automáticos ou semi-automáticos. Como o processo de classificação tem em conta o contexto em que o identificador é utilizado, mesmo que se utilizem termos diferentes para descrever o mesmo conceito estes serão armazenados em zonas adjacentes. Este facto facilita a inspecção manual pois existe o conceito de localidade.

3.1 Classificação

Ao olharmos de perto para as entidades utilizadas para representar a informação podemos concluir que, em última análise, usamos valores para descrever situações concretas e nomes. Os nomes são abstracções utilizadas para representar vistas, objectos, variáveis, funções, tipos de dados, etc. [6]. Mais importante é o facto de o nome representar o mesmo conceito desde a fase de requisitos até ao código. O facto de aparecerem novos nomes ou desaparecerem alguns dos existentes representa um indicador de possíveis desvios ao longo do processo de desenvolvimento. Os nomes podem pois ser entendidos como a assinatura do componente.

Neste trabalho concentramos a atenção nos nomes que se encontram associados a alguma informação, os identificadores, para representar o sistema. Se separamos os identificadores do resto da informação ficamos com dois domínios distintos: o domínio dos nomes e o domínio dos valores. Assumindo que o essencial da informação se situa no domínio dos nomes, podemos através de uma classificação estruturada, obter uma representação compacta e fácil de analisar que retenha as características essencias do componente.

No entanto, o identificador pode não ser só por si esclarecedor, mesmo que o nome que lhe está associado o seja. De igual forma a informação de contexto onde o identificador se situa pode não ser suficientemente esclarecedora. Para tal deve ser possível utilizar atributos que permitam esclarecer o comportamento ou a funcionalidade do identificador num dado contexto. Os atributos, apesar da sua função particular, podem também ser tratados como identificadores que caracterizam outros identificadores e cujo valor pode coexistir com os restantes valores.

Desta forma é possível obter um modelo que apresenta grande uniformidade e economia de conceitos. A uniformidade é útil pois é de mais fácil compreensão e manipulação, por não existirem casos especiais, quer para tratamento automático quer manual. A economia de conceitos permite obter grande expressividade e flexibilidade com um número reduzido de comportamentos distintos [17,18].

3.2 Representação

As hierarquias são estruturas organizadas onde diferentes níveis de abstracção podem ser manipulados a diferentes níveis da hierarquia. Os problemas são divididos em níveis de complexidade crescente [1]. O pormenor não desaparece, é empurrado para um nível mais baixo da hierarquia. Mais importante é o facto de a hierarquia ser uma forma de representação que surge naturalmente às pessoas, da mesma forma que é facil conceber ferramentas automáticas para a percorrer. Ao descrever um problema, quer numa perspectiva *bottom-up* ou *top-down*, a solução pode ser entendida como a raíz enquanto os problemas simples são representados nas folhas.

Os identificadores usados na descrição dos objectos são modelados numa àrvore, retendo uma referência para a sua posição original no objecto. Estes identificadores, depois de caracterizados, continuam a exibir as associações dos objectos originais [3]. As equivalências representam, numa segunda fase, essas mesmas associações na àrvore que se transforma num grafo [10]. São as equivalências que permitem descrever relações de partilha bem como definir objectos como extensões de outros [16]. Desta forma, enquanto os identificadores designam as entidades originais e os seus valores, as equivalências permitem a um identificador referir outro identificador.

Além dos identificadores e das equivalências que trabalham no domínio dos nomes existem ainda as referências e os tipos que trabalham no domínio dos valores. Uma referência permite a uma entidade referir outra e é por vezes designada por ponteiro. O tipo permite a uma entidade no domínio dos valores referir um identificador no domínio dos nomes. Quer as referências quer os tipos podem, normalmente, ser inferidos a partir dos dados originais. No entanto, caso estes estejam incompletos ou incorrectos é possível fazer alterações ao domínio dos valores sem alterar os documentos de onde a informação para os construir foi extraída.

4 Manipulação

Na secção anterior obtivemos uma descrição do sistema que representa os nomes das entidades numa àrvore. Componentes de dimensão média ou grande produzem, quando descritos em pormenor, grandes àrvores de nomes. Uma vez que o objectivo é obter descrições compactas e facilmente manipuláveis é necessário dispôr de um conjunto de mecanismos de selecção automática. Estes mecanismos permitem extrair àrvores constituídas por um sub-conjunto de nomes. A comparação entre várias soluções pode-se restringir a uma única vista. Por outro lado, a análise do contexto de um identificador em várias vistas permite retirar conclusões importantes sob a sua evolução no processo de desenvolvimento. Nesta secção serão focadas as técnicas de selecção, quer automática quer manual, com vista a permitir obter conclusões úteis do sistema.

4.1 Localização e Busca

Como o sistema é baseado na repetição de um conjunto de estruturas simples, com origem numa àrvore de nomes, as ferramentas são também simples e compactas. Embora as àrvores produzidas sejam à partida excessivamente grandes para permitir uma análise manual esclarecida, são pequenas dadas as velocidades de qualquer modesto computador pessoal. Para obter sub-àrvores de menores dimensões que permitam responder a perguntas específicas, são fundamentais três operações base: determinação

do contexto, qualificação do identificador e localização. Uma quarta operação de busca pode depois ser efectuada manualmente.

A busca numa àrvore é realizada usando dois graus de liberdade: largura e profundidade [22]. Se a àrvore for muito grande o processo torna-se moroso e, por vezes, inconclusivo. Ao extrair uma sub-àrvore de menores dimensões pode-se analisar pormenores de interpretação que dificilmente seria possível realizar com métodos totalmente automáticos.

Operações destinadas a obter o contexto de um identificador devolvem o conjunto de identificadores que constituem o caminho desde a raíz da àrvore até ao identificador em questão. Estas operações permitem construir uma nova àrvore em que o mesmo identificador surge em vários contextos, para os poder comparar. Alternativamente, podem-se seleccionar vàrios identificadores de uma mesma vista com o propósito de obter uma representação mais compacta da mesma. Uma operação de qualificação de um identificador permite determinar os seus atributos, ou seja, os identificadores do nível imediatamente abaixo daquele onde se situa. Desta forma pode-se enriquecer alguns identificadores com informação adicional em detrimento de outros menos importantes para uma dada perspectiva de análise. A operação de localização oferece o conjunto de identificador que existem no mesmo nível de um dado identificador, para um determinado contexto. Consegue-se, com esta operação, obter uma visão mais rigorosa dos identificadores com que este se relaciona e entre os quais existe, quase certamente, uma forte dependência.

Enquanto alguns modelos optam por oferecer um elevado número de construções e de operações [2], neste trabalho optou-se por oferecer um conjunto de primitivas que podem ser encadeadas. Desta forma, embora parte do trabalho possa recair sobre o utilizador, permite-se uma maior flexibilidade. Uma pequena linguagem de apoio permite definir sequências que podem ser posteriormente invocadas.

Uma comparação rude pode ser realizada se utilizarmos o comando *grep* do UNIX. Este comando proporciona-nos apenas a linha de texto onde o identificador se situa o que é, normalmente, manifestamente insuficiente. O sistema proposto, por outro lado, não retira muita da sintaxe da linguagem como agrupa as palavras chave que efectivamente permitem obter informação útil.

4.2 Processo de Selecção

O sistema até agora descrito tem que ser utilizado para produzir resultados úteis. O processo de selecção vai extrair um subconjunto de identificadores para responder a uma determinada dúvida do utilizador. Claro que podem ser necessárias várias iterações até que se atinjam conclusões fiáveis, mas as alternativas resumem-se frequentemente à análise do código propriamente dito. Evita-se, assim, ter de conhecer diversas linguagens de especificação e de programação, bem como os estilos de cada utilizador.

Nem toda a informação do documento original é convertida na representação de identificadores. Em primeiro lugar, parte da informação não está directamente associada aos identificadores, como é o caso de constantes. Por outro lado, o conversor utilizado pode não extrair a totalidade da informação, quer por limitações do conversor quer por opções de análise. Assim, poderá, por vezes, ser necessário recorrer ao código, mas apenas em situações extremas.

O domínio dos identificadores funciona como uma assinatura complexa do componente. Aliás esta técnica é uma extensão a métodos de assinaturas, que usam técnicas muito mais simples com base em menos informação. Contudo, estas assinaturas complexas conseguem reflectir muitas das características, quer estáticas quer dinâmicas, do componente que modelam.

Uma análise simples de contexto pode ser efectuada extraindo o conjunto de identificadores que localizam esse identificador numa vista específica. Se o identificador não for previamente conhecido, ou seja não tiver um nome bem conhecido, terá que ser efectuada uma busca nos seus atributos. A informação de contexto mostra essencialmente a posição relativa que o identificador tem no sistema na sua globalidade. Esta informação é útil para inferir qual é a aplicação em que o identificador está a ser utilizado, uma vez que o mesmo nome pode ser utilizado em várias situações.

Se extendermos o conceito anterior a várias vistas, ou conjuntos de contextos um para cada vista, obteremos um sub-conjunto de contextos para o mesmo identificador. Pode-se, assim, comparar alterações de ambiente de utilização de um dado identificador nas várias situações. Se estas várias vistas representarem várias fases do desenvolvimento, estas variações podem ser indicativas de um determinado conceito estar a ser utilizado com um objectivo numa fase inicial de especificação, mas à medida que o projecto evolui, ele vai sendo utilizado em situações diversas.

Quando o número de identificadores que nos conduzem desde a origem do sistema até um dado identificador, ou seja o caminho até esse identificador é bastante grande, temos uma informação detalhada sobre o seu contexto de utilização. Por outro lado, se esse caminho for curto a informação disponibilizada pode ser manifestamente insuficiente. Neste último caso poderá ser necessário utilizar operações adicionais para determinar o contexto de utilização do identificador. Uma operação que pode ser efectuada é a operação de qualificação, que permite obter um conjunto de atributos para esse identificador. Pode-se desta forma determinar através dos seus componentes qual a sua utilização. Uma outra forma de determinar complementarmente a informação de contexto usa uma operação de localização para obter o conjunto de identificadores que são usados no mesmo contexto. Estes identificadores podem permitir inferir através do ambiente qual a utilização dada ao identificador em questão. Se nenhuma desta informação for suficiente para determinar a funcionalidade do identificador então será necessário recorrer directamente ao documento. Este documento pode ser o código ou uma fase anterior de desenvolvimento, como a análise ou o desenho.

Além da informação de contexto poderemos obter informação mais complexa, nomeadamente recorrendo a comparações entre conjuntos de identificadores. A comparação dos atributos de um determinado identificador em dois contextos, por exemplo duas vistas, podemos determinar se alguma informação está a ser perdida ou adicionada. Neste último caso, é necessário verificar se o aumento do número de atributos contraria os requisitos ou não. É natural que com a evolução do sistema, e à medida que opções vão sendo tomadas, o número de atributos vá reflectindo esse enriquecimento por um aumento significativo do seu número. Note-se que o sistema só por si não permite uma determinação automática da correcção, mas deverá ser uma posterior inspecção manual que permitirá concluir se esses atributos podem ser considerados correctos. Este tipo de conclusões é extremamente difícil de introduzir em aplicações, mesmo recorrendo a mecanismos de inteligência artifical ou programação genética.

Este sistema não oferece resultados quantitativos, não é baseado em nenhum sistema de métricas. Conclusões que sejam tiradas com base na dimensão da àrvore gerada ou no comprimento do caminho de um identificador não representam métricas válidas. Estas observações podem depender de factores como estilo de programação ou a linguagem utilizada. Esta aproximação é baseada numa análise manual sendo precedida de um certo automatismo para seleccionar a informação relevante, evitando a consulta exaustiva dos documentos. Como qualquer sistema que seja dependente da experiência humana, o seu sucesso está directamente ligado com o treino e a capacidade de interpretação do engenheiro de software que o utiliza [14]. Por outro lado, oferece um grande manancial de informação de uma forma compacta e que pode ser manipulada de forma semi-automática.

5 Validação

Um teste simples de validação foi efectuado usando vários grupos de estudantes. A mesma especificação foi dada aos vários grupos, tendo cada grupo entregue uma especificação em UML [4,5] com uma representação gráfica das fases de análise e desenho. Posteriormente, entregaram uma solução em C++ [12,19] . O teste consistiu em duas partes. A parte de verificação de consistência procurou identificar inconsistências com a especificação. A parte de inter-operabilidade permitiu detectar qual a facilidade de trocar os módulos, entre as aproximações disponíveis. Da perspectiva dos estudantes estes testes representam uma medida da capacidade de interpretação da especificação e da modularidade do código produzido. O teste consistiu, numa primeira fase, na extracção das soluções de análise e desenho. Esta extracção foi efectuada manualmente, embora a linguagem UML tenha sido desenhada para permitir um processamento automático. Ou seja, pode ser compilada para efeitos de verificações de consistência. No entanto, ainda não foi escrito nenhum extractor para tal. A segunda fase da extracção do código, em C++, utilizou um *parser* simples que extrai apenas algumas

características da linguagem. Note-se que as àrvores obtidas a partir da extracção do C++ são muito maiores que as que têm origem em UML, mas a linguagem é especialmente complexa e modela uma fase final de desenvolvimento.

O primeiro resultado foi obtido pela análise directa da àrvore resultante. A dimensão das àrvores, embora não seja à partida uma métrica válida, permite especular quanto à complexidade e repetitividade da solução. Assim, àrvores anormalmente grandes estavam associadas a código de fraca qualidade enquanto àrvores muito reduzidas representavam, normalmente, o incumprimento total dos requisitos. No entanto, casos houve em que àrvores de pequena dimensão representavam soluções especialmente trabalhadas e de boa qualidade.

A localização de identificadores específicos foi extremamente importante para ter uma ideia precisa da funcionalidade. No entanto, determinar quais eram os identificadores mais importantes e ricos em informação pode não ser simples, excepto em casos frequentemente estudados. Neste caso, a experiência anterior permitia-nos saber de ante-mão quais seriam os pontos críticos. A sua análise foi, de facto, conclusiva para construção de uma visão global de cada solução apresentada. A escolha dos nomes dos identificadores verificou-se determinante no sucesso da metodologia. Verificou-se que algumas das soluções optaram por uma escolha especialmente infeliz de nomes, dificultando grandemente o processo de localização dos identificadores. Contudo, verificou-se que estas soluções estavam associadas a grupos de fraca qualidade técnica.

Esta aproximação provou ser especialmente útil nos casos bem comportados, que representam a maioria das soluções apresentadas. No entanto, a perspectiva pouco usual dos restantes casos ou era inconclusiva ou induzia conclusões menos correctas e precisas. Estes casos eram na sua quase totalidade soluções de fraca qualidade e que prestavam pouca atenção aos problemas de engenharia de software, nomeadamente aqueles que envolvem a reutilização.

6 Conclusões

A utilização de identificadores provou ser uma boa escolha, uma vez que oferece uma boa perspectiva do sistema. Por vezes existe a necessidade de inspecionar a informação original, mas não é expectável que qualquer abstracção possa substituir completamente a realização. A escolha criteriosa dos nomes dos identificadores apresentou-se como o ponto mais fraco do sistema, obrigando a um esforço adicional significativo. De notar que algumas das perspectivas mais simples existentes utilizam uma forma de catalogação em que as escolhas são limitadas eliminando quase totalmente este problema.

A escolha de uma hierarquia provou ser igualmente acertada pois permite de uma forma simples e intuitiva obter a informação da situação. A comparação dessa informação entre os vários casos permite obter respostas com significado real. Esta solução é também especialmente atractiva pois a maioria dos engenheiros de software está familiarizado com grandes estruturas de dados deste tipo como é o caso dos sistemas de ficheiros. A existência de vários níveis de profundidade e o significado físico do caminho da origem até um determinado ficheiro são analogias especialmente úteis na utilização do sistema. Uma representação gráfica da hierarquia poderá, contudo, vir a melhorar significativamente, no futuro, a sensibilidade e poder de assimilação.

O sistema proposto permite obter resultados conclusivos de uma forma rápida, uma vez que o processo de extracção pode ser feito de uma forma automática e a análise se limita a um sub-conjunto da informação extraída. No caso dos projectos serem excessivamente grandes a informação pode ser muito vasta para que uma primeira análise possa ser conclusiva, levando mais tempo a obter respostas úteis. No entanto, nestes casos a análise do código propriamente dito seria também muito mais morosa. Esperamos que a experiência na utilização do sistema permita identificar um conjunto de operações mais complexas, mas que permitam obter resultados mais rapidamente. No entanto, pretende-se que a análise não se baseie numa só busca mas que o engenheiro de software tenha a oportunidade de obter uma visão geral do componente por forma a realizar uma escolha mais esclarecida.

Bibliografia

.[1] Thomas Ball e Stephen G. Eick. Software visualization in the large. IEEE Computer, 29(4):33-43, Abril 1996.

[2] Sergio C. Bandinelli, Alfonso Fuggetta, e Carlo Ghezzi. Software process model evolution in the spade environment. IEEE Transactions on Software Engineering, 19(12):1128-1144, Dezembro 1993.

[3] Daniel Bardou e Christophe Dony. Split objects: a disciplined use of delegation within objects. In Object-Oriented Programming Systems and Applications, 1996.

[4] G. Booch, I. Jacobson, e J. Rumbaugh. The Unified Modeling Language for Object-Oriented Development. Rational Software Coporation, edição 0.91, Setembro 1996.

[5] G. Booch, I. Jacobson, e J. Rumbaugh. Unified Modeling Language Semantics. Rational Software Coporation, edição 1.0, Janeiro 1997.

[6] David Boundy. A taxonomy of programmers. Software Engineering Notes, 16(4):23-30, Outubro 1991.

[7] Greg Butler e Pierre Denommée. Documenting frameworks. Em 8th Annual Workshop on Software Reuse, Março 1997.

[8] Gianluigi Caldiera e Victor R. Basili. Identifying and qualifying reusable software components. IEEE Computer, 24(2):61-70, Fevereiro 1991.

[9] Pedro Reis Santos. Identifier based representation and management of software components. Em Workshop on Modeling Software Processes and Artifacts, 11ª ECOOP, Junho 1997.

[10] Jeffrey R. Van Dyke. Link Architecture for a Global Information Infrastructure. Tese de doutoramento, Massachutts Institute of Technology, Junho 1995.

[11] William B. Frakes e Christopher J. Fox. Sixteen questions about software reuse. Communications of the ACM, 38(6):75-87, Junho 1995.

[12] Stanley B. Lippman. C++ Primer. Addison-Wesley,Reading, MA, USA, 2ª edição, 1991.

[13] Steve McConnell. Keep it simple. IEEE Software, 13(11), Novembro 1996.

[14] Michael C. McFarland. The social implications of computarization: Making the technology more humane. Em 26ª ACM/IEEE Design Automation Conference, pages 129-134, 1989.

[15] Rubén Pietro'Diáz. Implementing faceted classification for software reuse. Communications of the ACM, 34(5):89-97, Maio 1991

[16] Hernán Astudillo R. Reorganizing split objects. Em Object-Oriented Programming Systems and Applications, 1996.

[17] Jerzy W. Rozenblit e Sanjaya Kumar. Toward synergistic engineering of computer systems. *IEEE Computer*, 30(2):126-127, Fevereiro 1997.

[18] António Rito Silva, Pedro Sousa e José Alves Marques. Development of distributed applications with separation of concerns. Em *Asia-Pacific Software Engineering Conference*, Digital Equipment Corporation 1995.

[19] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, MA, USA, 2ª edição, 1991.

[20] Anneliese von Mayrhauser e A. Marie Vans. Program comprehension during software maintenance and evolution. *IEEE Computer*, 28(8):44-55, Agosto 1995.

[21] Anthony I. Wasserman. Toward a discipline of software engineering. *IEEE Software*, 13(11), Novembro 1996.

[22] M. Wein, Wm Cowan e W. M. Gentleman. Visual support for version management. Em *Symposium on Applied Computing ACM/SIGAPP*, páginas 1217-1233, Março 1992.

[23] Pedro Reis dos Santos e Rui Gustavo Crespo. Assisted Selection of Components using Classified Identifiers. *7th Conference on Information Processing and Management of Uncertainty in Knowledged-based Systems*, 740-747, Julho 1998

[24] R. T. Mittermeir, H. Pozewauning, A. Mili e R. Mili. Uncertainty Aspects in Component Retrieval. *7th Conference on Information Processing and Management of Uncertainty in Knowledged-based Systems*, 564-571, Julho 1998

[25] A. M. Zaremski e J. M. Wing. Signature matching: a tool for using software libraries. *ACM Transactions on Software Engineering and Methodology*, 4(2):146-170, Abril 1995.

[26] R.T. Mittermeir e E. Kofler. Layered specifications to support reusability and integrability. *Journal of Systems Integration*, 3(3):273-302, Setembro 1993.