Visando o Reuso Formalmente Justificado de Especificações

João J. C. Gondim
Depto. de Ciência da Computação - Univ. de Brasília
e-mail: gondim@cic.unb.br

19 Maio 1995

Abstract

The traditional approach in software engineering is to reuse whole program modules, focusing on the code. Here, we depart from that approach in two ways: firstly by working on the specification; second, by attempting to reuse modules partially. Thus, a framework based on logic is proposed to address the problem, which is described in those terms. A solution based on software-engineering-sound reuse principles is obtained. This solution avoids some of the disadvantages other approaches experience. It was also possible to establish a link these results with others, related to the implementation process formalization and how to obtain simple formally justified reuse methodologies.

Resumo

As abordagens tradicionais de reuso em engenharia de software estão centradas na reutilização integral de módulos de programas a nível de código. Neste trabalho, nos distanciamos desta abordagem de duas formas: primeiro por trabalhar com especificações ao invés de programas; segundo por permitir o reuso parcial de módulos. Para isto, propomos uma framework para tratar do problema baseada na logica formal. Descrevemos o problema nestes termos e, baseando-se em princípios de reuso que fazem sentido em termos de engenharia de sofware, obtemos uma solução que evita algumas desvantagens de outras abordagens. Conseguiu-se tambem relcionar os resultados obtidos com outros referentes à formalização do processo de implementação, e ainda como é possível obter metodologias de reuso que são simples e formalmente justificadas.

1 Introdução

Hoje a questão da reutilização de software é central à engenharia de software. Freqüentemente, esta é tratada sob a ótica do reaproveitamento integral de módulos de um sistema. A ênfase desta abordagem normalmente recai no reuso de código.

Neste artigo, é proposta uma *framework* para tratar formalmente do problema do reuso que se afasta da abordagem citada de duas formas. Primeiro, por dar ênfase ao reuso da especificação ao invés do código. Segundo por não supor o reuso integral de especificações de módulos, mas parcial.

A primeira destas é de motivação metodológica, viabilizando uma abordagem formal da questão. A outra por se vislumbrar o suporte formal de atividades relevantes do processo de desenvolvimento formal de software.

A reutilização de partes de uma especificação normalmente acontece quando da construção da especificação e da evolução do aplicativo. Estas atividades em geral são executadas de maneira *ad hoc*. Serão propostas formas de suportar estas atividades formalmente.

Este artigo está organizado em seções. Na que segue, será apresentada a framework proposta, com comparações com outras. Depois, serão apresentadas alguns propriedades formais da mesma que suportam formalmente algumas práticas comuns de engenharia de software. O artigo conclui fazendo a conexão com trabalhos correlatos e apontando caminhos para exploração dos resultados aqui apresentados.

2 Reuso via revisão

A reutilização é orientada à especificação. Especificações serão tratadas como apresentações de teorias, *i.e.* teorias em linguagens de primeira ordem poli-sortidas, ou não, definidas por um conjunto finito de axiomas, como em [Turski e Maibaum 88].

Antes de partir para o caso do reuso, vejamos a sua motivação. Seja $\mathcal{T}$ uma

teoria suposta consistente e possivelmente incompleta, e uma sentença $\alpha \in \mathcal{L}(\mathcal{T})$, a linguagem de $\mathcal{T}$. α seria uma expressão que se deseja conste da versão final (ou da *próxima* versão) da especificação. Quer se obter uma teoria $\mathcal{T}'$, onde $\mathcal{L}(\mathcal{T}') = \mathcal{L}(\mathcal{T})$ tal que:

- $\alpha \in \mathcal{T}'$
- $\mathcal{T} \cap \mathcal{T}' \neq \emptyset$
- $\mathcal{T}'$ é consistente.

temos três possibilidades:

- $\alpha \in \mathcal{T}$
- $\alpha \notin \mathcal{T}$ e $\neg\alpha \notin \mathcal{T}$
- $\alpha \notin \mathcal{T}$ e $\mathcal{T} \cup \{\alpha\} \vdash \bot$

o primeiro caso é menos que trivial:

$$\mathcal{T}' = \mathcal{T}$$

No segundo, $\mathcal{T}'$ é uma extensão não conservativa de $\mathcal{T}$:

$$\mathcal{T}' = \mathcal{T} \cup \{\alpha\}$$

no último caso ocorre a necessidade de uma revisão da teoria $\mathcal{T}$. $\mathcal{T}'$ seria então α junto com a parte de $\mathcal{T}$ que é consistente com α. O problema agora é encontrar esta parte de $\mathcal{T}$ que é consistente com α.

Ocorre entretanto que fica intratável lidar com uma teoria $\mathcal{T}$ toda. Como uma teoria é gerada de uma apresentação, e esta é finita, o problema será atacado por este lado. Esta decisão vem de encontro ao que seria razoável num contexto de engenharia de software, posto que especificações, por mais longas que sejam, devem ser objetos finitos.

Em [Gärdenfors 88], uma abordagem de mínima contração é sugerida para a revisão de teorias. Esta porém leva a teorias completas, i.e., ainda que a teoria original seja incompleta, a teoria revisada é completa. De certa forma, a teoria revisada é maior do que deveria ser. Do ponto de vista de engenharia de software, não é interessante ter uma teoria completa, já que a esta corresponderia uma especificação completa, onde não cabem refinamentos.

Visando contornar as dificuldades acima [Gärdenfors 88] sugere a introdução de priorização das sentenças da linguagem, o que evitaria não só o problema da completude compulsória mas também da adequação da teoria revisada, no sentido em que esta não consegue preservar tudo que se desejava da teoria original. [Ryan 92] operacionaliza a proposta sugerindo apresentações de teorias estruturadas, onde a estruturação seria feita na forma de ordenação das sentenças da linguagem, sendo daí derivadas as noções de satisfação, modelos, etc. A ordem seria um espelho da importância relativa das sentenças da linguagem na organização do conhecimento da teoria.

Esta última proposta apresenta uma revisão em alguns aspectos mais próxima do que seria intuitivo, porém peca por imunizar a teoria revisada contra possíveis contradições. A principal restrição a esta abordagem seria a introdução de todo um aparato formal destinado não à representação do conhecimento sobre a aplicação mas sobre a forma como a formalização foi feita. Exige-se assim não apenas a formalização do conhecimento acerca da aplicação em si, mas também acerca do processo de formalização. Do ponto de vista da utilização, fica a questão de como ordenar a teoria, o que acaba sendo realizado de maneira *ad hoc*. Na verdade, exigir do usuário tal nível de internalização acerca do conhecimento que ele encapsula em uma teoria é no mínimo irrealístico. A título de exemplo, sugerimos que se tente ordenar, segundo a importância dos axiomas, uma teoria simples e bem conhecida como a que segue (com símbolos extra-lógicos 0 e *suc*, e) apresentada pelos axiomas:

Ax1 . $\forall x.\neg suc(x) = 0$

Ax2 . $\forall x \forall y.suc(x) = suc(y) \rightarrow x = y$

Ax3 . $\forall y.(y \neq 0 \rightarrow \exists x.y = suc(x))$

como se vê, apesar de haver várias maneiras de realizar tal ordenação, é difícil que uma delas seja formalmente justificada. Assim, seguramente a ordenação irá introduzir graus de incerteza quando da construção da especificação.

3 Reusando a apresentação

Como já se disse, a tentativa de operar a revisão trabalhando com a teoria inteira resulta não apenas impraticável bem como contra intuitiva. Assim, o foco de atenção da revisão recai sobre a apresentação da teoria, lembrando que é no contexto da revisão que será abordada a questão do reuso.

Definição 3.1 *Um conjunto de sentenças Ψ é dito ser consistente com respeito a outro conjunto Φ sss $\Psi \bigcup \Phi$ é consistente.*

o

Uma das dificuldades encontradas na abordagem em [Gärdenfors 88] no uso de uma estratégia de contração mínima estava no fato das teorias revisadas serem completas. Isto ocorre porque o conceito de contração mínima é baseada no conceito de consistência maximal.

Como já se disse esta completude compulsória da teoria revisada é indesejável. Não apenas por levar a teorias maiores do que deveriam ser, mas também por não fazer muito sentido em um contexto de desenvolvimento de software.

A questão chave parece estar no fato de que, intuitivamente, a revisão envolve dois conjuntos de sentenças: um que motiva a revisão e outro em que esta será executada. A idéia é maximizar o que é preservado do último, de forma que possa

ser expandido com o primeiro e resultar consistente. É deste último que as sentenças devem ser escolhidas e a escolha deve ser maximal com respeito ao último conjunto.

Como se sabe, a definição de consistência maximal tem como conseqüência que os conjuntos de sentenças com tal propriedade são completos. Evitar esta completude é a motivação para a definição de consistência maximal de conjuntos de sentenças, apresentada a seguir, que difere da usual (vide [Enderton 72]).

Definição 3.2 *Seja Ψ um conjunto de sentenças e um subconjunto Ψ_0 de Ψ, $\Psi_0 \subseteq \Psi$. Ψ_0 é dito ser um subconjunto maximal consistente de Ψ sss este é consistente e para qualquer outro subconjunto $\Psi' \subset \Psi$, tal que $\Psi_0 \subset \Psi'$, então Ψ' é inconsistente.*

o

Note-se que esta definição recai na usual se $\Psi = \mathcal{L}(\Psi_0)$. Mais, relativizando a consistência maximal a um conjunto genérico de sentenças, não segue necessariamente que o (sub) conjunto maximal consistente é completo, como no caso da definição usual.

A próxima definição relativiza o conceito a outro conjunto.

Definição 3.3 *Sejam Φ e Ψ conjuntos de sentenças, e um subconjunto Φ_0 de Φ, $\Phi_0 \subseteq \Phi$. Φ_0 é dito ser maximal consistente com Ψ se ambos são consistentes e se não existe Φ_1, $\Phi_1 \subseteq \Phi$, consistente com Ψ tal que: $\Phi_0 \subset \Phi_1$.*

o

Antes de apresentar os postulados, ou princípios de revisão, lembramos que a motivação é ver a revisão como a restauração de consistência em face à nova informação que deve ser incorporada ao modelo da aplicação correspondente à teoria.

Definição 3.4 *Seja Θ uma apresentação para uma teoria $\mathcal{T}$, Δ um conjunto de sentenças tal que $\Theta \vdash_{\mathcal{T}} \Delta$ e Γ um conjunto de sentenças, todos eles em $\mathcal{L}(\mathcal{T})$. Uma*

revisão de T é outra teoria T' com apresentação Θ', tal que:

$$\Theta' \vdash_{T'} \Gamma \ e,$$
$$\Theta' \vdash_{T'} \Delta'$$

onde $\Delta' \subseteq \Delta$ e é maximal.

o

O conjunto de sentenças que motiva a revisão será chamado de conjunto de revisão, ou revisor. A teoria revisada é obtida priorizando as sentenças no conjunto de revisão e um complemento que deve satisfazer as condições que seguem. Seja $\Theta' = \Theta_0$, onde Θ_0 satisfaz os postulados:

Reutilização da apresentação

$$\Theta_0 \subseteq \Theta$$

a apresentação da teoria revisada deve se basear na apresentação original.

Consistência da apresentação com respeito ao conjunto revisor

$$\Theta_0 \text{ consistente com } \Gamma$$

o reusável da apresentação, Θ_0, deve se conformar ao revisor Γ.

Priorização da informação do revisor

$$\Gamma, \Theta_0 \vdash_{T'} \Delta_0$$

normalmente as informações expressas pelas sentenças do revisor Γ são de motivação factual, enquanto as de Θ têm caráter conjectural, priorizando os fatos na revisão, a nova apresentação é obtida juntando-se ao revisor o que se reusou da teoria original.

Persistência conseqüêncial

$$\Delta_0 \subseteq \Delta$$

a teoria revisada compartilha conseqüências com a original.

Consistência conseqüêncial com respeito ao revisor

Δ_0 maximal consistente com Γ

a teoria revisada se conforma ao revisor.

Os postulados acima tentam capturar a intuição da situação em que um engenheiro de software que tem diante de si uma lista (presumivelmente longa) de axiomas de uma especificação e outra lista de revisões a estes axiomas. A primeira tarefa que deverá ser executada será identificar na lista de axiomas quais serão alterados e quais não serão alterados. Estes últimos serão a porção reutilizavel da especificação. Os outros também se dividem em dois grupos: o dos que serão descartados e os que serão alterados. A estes todos junta-se a lista das revisões.

Diante da motivação acima, será apresentado a seguir um resultado que sugere como viabilizar esta separação dos axiomas da especificação original. Será mostrado também que o formato da apresentação terá influência no resultado da separação.

Quanto à questão das correções, serão indicadas no final formas de tratá-las. Entretanto, adiantamos que, intuitivamente, parte delas já está disponível na forma do próprio conjunto revisor em si, restando realizar as alterações nos axiomas assinalados para tal.

Os postulados acima levam a uma forma de revisão que não tem o problema da completude compulsória, porém pode levar a uma teoria menor que o desejado. Isto se deve às opções iniciais de se atacar a apresentação; e de fazê-lo sem a introdução de aparatos formais estranhos á lógica em si. Em termos da questão do reuso, será visto que os postulados acima levam a uma forma simples e segura de realizá-lo.

4 Revisão e Interpolação

Seja T' uma teoria com apresentação Θ'. Considere $\Theta' = \Theta_0$, onde Θ_0 satisfaz os postulados da seção anterior:

$$\Theta_0 \subseteq \Theta$$

$$\Theta_0 \text{ consistente com } \Gamma$$

$$\Gamma, \Theta_0 \vdash_{T'} \Delta_0$$

$$\Delta_0 \subseteq \Delta$$

$$\Delta_0 \text{ max. cons. } \Gamma$$

Sejam Γ, Θ, Θ_0, Δ e Δ_0 conjuntos de sentenças satisfazendo as condições acima, Então Γ é dito ser o revisor, Θ o revisando, Θ_0 o revisado, Δ a conseqüência e Δ_0 a conseqüência revisada.

Desta forma, a apresentação da nova teoria (revisada) tem propriedades interessantes.

Proposição 4.1 *Se* $\Theta_0 \vdash_{T'} \alpha$, *então* $\alpha \in \Delta_0$.

Definição 4.1 *Sejam* Φ *e* Ψ *conjuntos de sentenças, cujas sentenças são respectivamente formadas pelos símbolos extra-lógicos nos conjuntos* Σ_Φ *e* Σ_Ψ *respectivamente, a linguagem comum deles é o conjunto de símbolos* $\Sigma_\Phi \bigcap \Sigma_\Psi$.

o

Definição 4.2 *Um conjunto de sentenças é dito estar na linguagem comum de outros dois conjuntos de sentenças se os símbolos que ocorrem em suas sentenças é um subconjunto da linguagem comum dos outros dois conjuntos.*

o

Proposição 4.2 Θ_0 *está na linguagem comum de* Θ *e* Δ_0.

Corolário 4.1 $\Sigma_{\Theta_0} \subseteq \Sigma_{\Delta_0}$

Proposição 4.3 *Θ_0 é maximal consistente com Γ.*

Proposição 4.4 *Se $\alpha \in \Delta_0$, então $\Theta_0 \vdash_{\mathcal{T}} \alpha$.*

Proposição 4.5

$$\Theta_0 \vdash_{\mathcal{T}} \Delta_0$$

Definição 4.3 (Interpolante de Craig) *Dados dois conjuntos de sentenças Γ e Δ, onde*

$$\Gamma \vdash \Delta$$

então, o conjunto Υ tal que $\Gamma \vdash \Upsilon$ e $\Upsilon \vdash \Delta$ e Υ está na linguagem comum de Γ e Δ é um conjunto de interpolantes de Craig (ou interpolantes) de Γ e Δ.

o

Lema 4.1 *Θ_0 é um conjunto de interpolantes de Θ e Δ_0.*

Corolário 4.2 *Dados conjuntos de sentenças Θ, Δ e Γ, cada um individualmente consistente, se existem conjuntos Θ_0 e Δ_0 satisfazendo os postulados de revisão, então existe um conjunto de interpolantes entre Θ e Δ_0.*

O conceito a seguir será instrumental para o reuso.

Definição 4.4 *Seja Θ uma apresentação de uma teoria $\mathcal{T}$, e Γ e Δ conjuntos de sentenças. A Separação de Θ, $S(\Theta)$, é uma classe formada pelos subconjuntos Θ_+, Θ_- e $\Theta_?$ de Θ onde:*

$$\Sigma_{\Theta_+} = \Sigma_\Delta - \Sigma_\Delta \bigcap \Sigma_\Gamma$$

$$\Sigma_{\Theta_-} = \Sigma_\Gamma$$

$$\Theta_? = \Theta - \Theta_+ \bigcup \Theta_-$$

o

E o resultado principal para o reuso é o que segue. Sendo o reusável a parte da apresentação que é reaproveitada, o descartável a parte que não é reusável e o alterável a parte que pode ser reusável sujeita a modificações, tem-se:

Lema 4.2 (Lema da Separação) *Seja Θ uma apresentação de uma teoria $\mathcal{T}$, e Γ e Δ_0 conjuntos de sentenças. Se Θ_0 é obtido segundo os postulados da revisão, então existe uma separação de Θ, $S(\Theta)$, em que*

$$\Theta_0 = \Theta_+$$

Assim, Θ_- será o descartável de Θ e $\Theta_?$ será o alterável de Θ.

O resultado acima provê assim uma forma simples de se identificar o que pode ser reusado, ou não, em uma especificação. E também o que deve ser alterado.

O resultado acima sugere o seguinte procedimento:

Entrada: a apresentação Θ, e um conjunto de revisões Γ_0, com símbolos extra-lógicos Σ_Γ.

Saída: $\Theta_0 = \Theta_+$, o reusável de Θ.

Faça: calcule a separação de Θ usando Γ_0 como Γ, o revisor.

Deve-se notar que o cálculo da separação envolve apenas uma inspeção sintática dos símbolos presentes nos axiomas em Θ, classificando-os segundo a presença dos símbolos Σ_Γ. Isto pode ser feito de forma extremamente simples e eficiente.

No exemplo a seguir será ilustrado como isto é feito.

Exemplo 4.1 *Considere a apresentação da teoria com símbolos extra-lógicos C, H, T e λ:*

Ax1 . $\forall x \forall y \forall w \forall z.(C(x,y) = C(w,z)) \equiv ((x = w) \wedge (y = z))$

Ax2 . $\forall x \forall y.H(C(x,y)) = x$

Ax3 . $\forall x \forall y.T(C(x,y)) \doteq y$

Ax4 . $T(\lambda) = \lambda$

Suponha que

$$\Sigma_\Gamma = \{\lambda\}$$

Desta forma Θ_0 *incluirá os três primeiros axiomas.*

•

Exemplo 4.2 *Considere a mesma teoria do exemplo acima. Se,*

$$\Sigma_\Gamma = \{\lambda, T\}$$

tem-se agora que Θ_0 *incluirá os dois primeiros axiomas. O último não é reusável e o terceiro é alterável.*

•

Exemplo 4.3 *Considere a teoria dos exemplos anteriores, porém com a seguinte apresentação:*

$$(\forall x \forall y \forall w \forall z.(C(x,y) = C(w,z)) \equiv ((x = w) \wedge (y = z))) \wedge$$
$$(\forall x \forall y.H(C(x,y)) = x) \wedge$$
$$(\forall x \forall y.T(C(x,y)) = y) \wedge$$
$$(T(\lambda) = \lambda)$$

Neste caso, para os revisores dos exemplos 4.1 e 4.2, em ambos os casos, não será possível reaproveitar o único azioma da apresentação, este porém poderá ser reaproveitado se alterado.

•

Este último exemplo indica que apesar de se ter a mesma teoria, certos formatos da apresentação da mesma podem facilitar a tarefa de separar a especificação, enquanto outras simplesmente inviabilizam toda a empreitada. Este fato estabelece um intrigante paralelo com programas, que quando sujeitos a certas disciplinas de

formatação de código se mostram mais susceptíveis à modificação. No caso específico do exemplo apresentado, seria o caso de se dizer que a apresentação da teoria em questão é menos legível e estruturada que a anterior, indo se encontro ao que normalmente não é recomendável. Fica assim a motivação para se buscar disciplinas de apresentação que propiciem a reusabilidade das mesmas. É interessante notar que [Poubel e Veloso 93] justifica levantar formalmente a questão do formato da especificação.

Outro ponto que se deve tocar, é a questão de como operar as alterações nos axiomas candidatos a tal. Primeiramente, pode-se voltar à questão dos formatos de apresentação e dizer que estes devem trabalhar na direção de minimizar o alterável (no caso ideal $\Theta_? = \emptyset$). No caso de ainda haver a necessidade de alterações, é bom lembrar que os resultados aqui apresentados exploram apenas a parte meramente sintática da propriedade de interpolação de Craig. Usando-se sua parte dedutiva, acredita-se poder chegar a procedimentos para efetuar as alterações.

5 Conclusões

Partindo da abstração em que uma especificação formal é tratada como uma teoria em uma linguagem lógica, foi possível propor uma *framework* para tratar da questão da reutilização de software, sendo esta vista como o reuso de partes da especificação. O reuso foi encaixado no contexto da revisão, porém seguindo uma motivação mais próxima do contexto de engenharia de software.

Os resultados obtidos, levam a uma maneira simples e eficiente de tratar do reuso da especificação. Entretanto, algumas questões pertinentes foram levantadas, que apesar de ter-se indicado a possível direção para atacá-las, ficam para os desdobramentos futuros deste trabalho.

Fica a questão dos formatos de apresentação que favorecem o reuso, e outra que lhe é complementar: a das alterações dos axiomas. Lembre-se porém que os

resultados não exploraram o lado dedutivo da propriedade de interpolação de Craig. É de se esperar que resultados nesta direção sejam obtidos.

Uma questão mais intrigante parece estar no fato da propriedade de interpolação de Craig, também estar presente no contexto da teoria da implementação. Fica a tentação de especular que relações formais reuso e implementação teriam, e até vislumbrar a possibilidade de um *framework* maior suportando uma visão global de processo de desenvolvimento de software toda baseada em lógica.

Referências

[Enderton 72] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[Turski e Maibaum 88] W. M. Turski e T. S. E. Maibaum. *The Specification of Computer Programs.* Addison-Wesley Pub. Co., 1988.

[Gärdenfors 88] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States.* The MIT Press, 1988.

[Ryan 92] M. Ryan. *Ordered Presentatons of Theories. PhD Thesis*, Imperial College of Science Technology and Medicine, 1992.

[Poubel e Veloso 93] H. W. Poubel e P. A. S. Veloso. *Sobre o Teorema da Modularização: Importância e uma Prova por Quociente*, in Anais do VII Simp. Bras. de Engenharia de Software, 1993.