Avaliação Experimental da Relação entre Coesão e o Esforço de Compreensão de Programas: Um Estudo Preliminar

Elienai B. Batista[1], Claudio Sant'Anna[1]

[1]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA) – Salvador – BA - Brasil

elienaibittencourt@gmail.com, santanna@dcc.ufba.br

Abstract. *The software engineering literature claims that cohesion is a software design attribute that influences program comprehensibility. Researchers have defined a series of metrics for quantifying cohesion. However, there is no empirical evidence about whether there is a relation between cohesion, quantified by means of metrics, and program comprehension effort. In this paper, we present a quasi-experiment that evaluates the relation between cohesion measures and the effort for understanding the source code of object-oriented systems' classes.*

Resumo. *A literatura de engenharia de software afirma que coesão é um atributo de design do software que influencia a compreensibilidade de programas. Pesquisadores definiram uma série de métricas para quantificar coesão. No entanto, faltam evidências experimentais sobre a existência de uma relação entre coesão, quantificada por meio de métricas, e o esforço para se compreender programas. Nesse artigo, apresentamos um quase-experimento que avalia a relação entre medidas de coesão e o esforço para se compreender o código fonte de classes de sistemas orientados a objetos.*

1. Introdução

Coesão é um conceito bem conhecido e usado como atributo de qualidade interna de *design* do software. A coesão de um módulo corresponde ao grau pelo qual o módulo se dedica a implementar apenas uma responsabilidade do sistema [Pflegger and Atlee 2010]. Afirma-se que um software bem projetado tem os módulos bem coesos. Afirma-se também que o grau de coesão dos módulos de um sistema pode influenciar atributos de qualidade externa importantes, como manutenibilidade, facilidade de compreensão e facilidade de reutilização [Pflegger and Atlee 2010].

Várias métricas tem sido definidas e utilizadas para quantificar valores de coesão de software orientado a objetos [Chidamber & Kemerer 1994; Henderson- Sellers et al. 1996; Briand et al. 1998; Silva et al. 2012]. A maioria dessas métricas quantifica coesão de cada uma das classes que compõem o código fonte do sistema. Essas métricas são de dois tipos: estruturais e conceituais. As métricas estruturais baseiam-se nas relações de dependência estrutural entre os métodos da classe para quantificar coesão [Briand et al. 1998]. Esse tipo de métrica considera que uma classe é bem coesa se a maioria dos seus métodos dependem um do outro ou acessam pelo menos um mesmo atributo da classe. Por outro lado, as métricas conceituais baseiam-se em informações dos conceitos implementados pelos métodos e nas relações de dependência conceitual entre eles. Se a maioria dos métodos implementam os mesmos conceitos, a classe é considerada bem

coesa. A maioria das métricas conceituais usa técnicas de mineração de textos para determinar os conceitos implementados por cada método [Marcus & Poshyvanyk 2005; Silva et al. 2012].

Acredita-se que, quanto maior for coesão, menor pode ser o esforço para se compreender um programa [Pflegger and Atlee 2010]. Módulos com baixa coesão são, teoricamente, mais difíceis de compreender, pois possuem código fonte relativo a diferentes responsabilidades, o que pode atrapalhar o entendimento de cada uma delas.

Compreensão de programa consiste da realização de atividades para se obter conhecimento geral sobre o código fonte de um sistema, as funcionalidades que ele implementa e como ele está estruturado [Bois et al. 2006]. Estima-se que desenvolvedores dediquem em média mais da metade do esforço de manutenção de software com atividades de compreensão [Rugaber 1995]. Além disso, parte substancial do esforço de compreensão de um sistema está relacionada à compreensão do seu código fonte.

Apesar de existir uma série de métricas de coesão e de se acreditar que coesão influencia a compreensibilidade de programas, poucos estudos foram realizados para avaliar qual é a relação entre coesão, quantificada por meio de métricas, e o esforço para se compreender o código fonte de sistemas de software. Diante desse contexto, realizamos um estudo preliminar com o objetivo de avaliar experimentalmente em que nível o grau de coesão de classes de sistemas de software orientados a objetos está relacionado ao esforço para compreender seu código fonte. Avaliamos também se os diferentes tipos de métricas – estrutural e conceitual – tem relação diferente com o esforço de compreensão. No estudo, participantes executaram tarefas de compreensão do código fonte de classes com diferentes graus de coesão conceitual e estrutural, quantificados por meio de métricas de código fonte. Os resultados trouxeram evidência da dificuldade de se analisar isoladamente o impacto de coesão, um vez que outros atributos do código fonte também podem influenciar a compreensibilidade. Por causa disso, os resultados não forneceram evidências conclusivas a respeito da relação entre coesão e os esforço para se compreender programas. Por outro lado, o design do estudo se mostrou promissor, principalmente no que tange a medição do esforço de compreensão de programas.

2. Trabalhos Relacionados

Os trabalhos relacionados, em sua maioria, ou (i) dedicam-se a avaliar a relação entre outras características do código fonte e compreensibilidade de programa, como Feigenspan et al. (2011), ou (ii) estudam a relação entre coesão com outros atributos de qualidade, como a tendência a mudanças ou a tendência a defeitos, como Silva et al. (2012). Não encontramos nenhum trabalho que avalie explicitamente se medidas de coesão estão associadas ao esforço para se compreender programas.

Feigenspan et al. (2011), por exemplo, analisam se e em que grau medidas de complexidade e tamanho, por exemplo, número de linhas código, estão relacionadas a compreensão do programa. Seus resultados mostram que não há correlação entre essas características do código fonte e a compreensão do programa. Os autores acreditam que possa existir uma relação entre coesão e acoplamento com a compreensão de programa e por isso recomendam uma investigação dessa relação. Silva et al. (2012), por outro lado, analisam a relação entre coesão com a tendência que as classes tem de sofrer mudanças. Eles avaliam a correlação estatística entre uma série de métricas de coesão e o número de

revisões (commits) que classes sofreram ao longo da evolução de alguns sistemas. Os resultados evidenciaram a existência de uma correlação positiva moderada entre algumas métricas de coesão e o número de revisões das classes.

3. Design do Estudo

O estudo experimental teve como objetivo responder as seguintes questões de pesquisa: **Questão 1:** A coesão de uma classe apresenta alguma influência sobre a compreensão do seu código?

Questão 2: Há diferença no esforço de compreensão de classes com diferentes valores de coesão conceitual e coesão estrutural?

O estudo teve a seguinte configuração: No laboratório, dezoito participantes realizaram atividades que demandaram a compreensão do código fonte de quatro classes. A realização dessas atividades permitiu que medíssemos o esforço desprendido por cada participante para compreender cada classe. As classes tinham diferentes valores de coesão para que pudéssemos compará-los com o esforço de compreensão. O estudo se caracterizou por ser preliminar devido pequena quantidade de participantes e classes envolvidas. Além disso, podemos classificar o estudo como um quase-experimento, pois os participantes foram selecionados por conveniência. A seguir mais detalhes do design do estudo são descritos.

Métricas: Utilizamos a métrica de coesão estrutural *Lack of Cohesion in Methods 5* (LCOM5) [Henderson- Sellers et al. 1996] e a métrica de coesão conceitual *Lack of Concern-Based Cohesion* (LCbC) [Silva et al. 2012]. Selecionamos LCOM5 por se tratar da versão mais recente da tradicional métrica de coesão LCOM [Chidamber & Kemerer 1994]. Para quantificar coesão, LCOM5 considera a dependência entre métodos por meio do acesso a atributos em comum. Quanto maior a quantidade de métodos que acessam atributos em comum, maior é a coesão da classe e vice-versa. Os valores de LCOM5 variam de zero a um. Quanto menor o valor de LCOM5 mais coesa é a classe. Portanto, se uma classe tem valor de LCOM5 igual a zero significa que todos os seus métodos acessam os mesmo atributos e, portanto, são bem coesos entre si. No estudo, calculamos os valores de LCOM5 por meio da ferramenta *Metrics*[1].

Selecionamos a métrica conceitual LCbC por ela já ter sido usada em outros estudos que a comparam com métricas estruturais [Silva et al. 2012; Silva et al. 2014]. LCbC conta o número de interesses presentes em cada classe [Silva et al. 2012]. Interesses são conceitos implementados em um sistema, como requisitos, funcionalidades e regras de negócio. Quanto mais interesses uma classe implementa, menos coesa ela é, e vice-versa. Para quantificar essa métrica é preciso determinar quais interesses cada método de uma classe implementa [Silva et al. 2012]. Isso pode ser feito manualmente ou usando alguma técnica de mineração de texto, como acontece com outras métricas conceituais [Liu, Y. et al]. Devido o pequeno número de classes, foi possível determinar manualmente os valores de LCbC para as classes do estudo.

Seleção de Classes: Essa foi a etapa mais crítica do estudo, pois ao selecionar as classes teríamos que minimizar ao máximo a influência de fatores que também pudessem influenciar coesão (fatores de confusão). Selecionamos, portanto, classes com as

[1]http://metrics.sourceforge.net/

[2] http://guimteixeira.wordpress.com/2010/05/16/exercicio-de-java-agenda-telefonica/

seguintes características: (i) tamanhos semelhantes em termos do número de linhas de código, (ii) domínio simples e acessível a todos os participantes, (iii) nomenclatura de identificadores com qualidade semelhante, (iv) e ausência de comentários. Além desses fatores, outro aspecto importante para a seleção das classes foram os valores das métricas LCOM5 e LCbC. A Tabela 1 apresenta os valores de coesão das classes selecionadas. Para permitir a comparação entre diferentes valores de coesão estrutural e conceitual, selecionamos: (i) uma classe com alta coesão estrutural e alta coesão conceitual (Contatos[2]), (ii) uma classe com baixa coesão estrutural e baixa coesão conceitual (Locadora2[3]), (iii) uma classe com alta coesão estrutural e baixa coesão conceitual (BibliotecaUI2[4]) e (iv) uma classe com baixa coesão estrutural e alta coesão conceitual (Person2[5]).

Tabela 1. Valores das métricas das classes selecionadas

Métrica	BibliotecaUI2	Person2	Locadora2	Contatos
LCOM5	0	0,93	0,6	0
LCbC	4	1	4	1
Número de Linhas (LOC)	274	254	240	200

Medição do esforço de compreensão: Para quantificar o esforço necessário para compreender cada classe usamos duas métricas: (i) o tempo utilizado pelos participantes para responder quatro perguntas sobre cada uma das classes e (ii) o número de perguntas com respostas erradas. As perguntas apresentadas nos questionários exigiam uma simulação mental do código fonte, como por exemplo, "Se o método *m* receber como entrada os valores *x* e *y*, qual será seu retorno?" Essa estratégia está alinhada com o que a literatura recomenda para se medir esforço de compreensão [Bois et al. 2006]. Segundo Dunsmore and Roper (2013) esse tipo de pergunta reflete melhor a compreensão do participante acerca do código fonte.

Participantes: A amostra foi composta por 18 participantes: 14 alunos de graduação e quatro alunos de pós-graduação em Ciência da Computação da Universidade Federal da Bahia. Aplicamos um questionário para verificar a experiência dos participante. De acordo com as respostas, consideramos sete participantes como experientes (mais de três anos trabalhando com programação) e 11 pouco experientes.

Estudo Piloto: Para avaliar a configuração do experimento realizamos um estudo piloto com dois participantes, um experiente e outro pouco experiente. Por meio do estudo piloto, verificamos que a execução do experimento duraria em média 50 minutos. Além disso, corrigimos alguns erros ortográficos encontrados nos questionários e verificamos que alguns esclarecimentos deveriam ser comunicados aos participantes antes da execução do experimento.

3.1. Ameaças à Validade

Abaixo apresentamos possíveis ameaças à validade do experimento e as ações realizadas para minimizá-las.

---

[2] http://guimteixeira.wordpress.com/2010/05/16/exercicio-de-java-agenda-telefonica/

[3] http://ifpr2011.blogspot.com.br/2011/09/locadora-em-java-usando-modo-texto.html

[4] http://www.robsonmartins.com/inform/java/persistencia.php

[5] http://www.soberit.tkk.fi/mmantyla/ISESE2006/

Validade Interna: A fadiga dos participantes ao analisar o código fonte é uma possível ameaça à validade interna. Para minimizar a influência da fadiga, configuramos o estudo de forma que sua execução não ultrapassasse uma hora de duração. Participantes com diferentes níveis de experiência em programação poderiam ser outra ameaça. Verificamos a experiência de cada participante, por meio de um questionário de caracterização, e observamos que a diferenças em graus de experiência não afetaram os resultados. Também nos preocupamos com o fato da falta de similaridade entre as classes poder influenciar os resultados. Tentamos minimizar essa ameaça levando em consideração alguns fatores de confusão ao selecionar as classes.

Validade Externa: O estudo possui a limitação de não poder ser generalizado devido à pequena quantidade de classes e à pequena amostra de participantes, composta somente por estudantes. Outra ameaça está relacionada ao fato dos participantes terem usado um editor de texto (WinEdit) mais limitado que os ambientes de programação comumente usados em ambientes profissionais. Fizemos essa opção de propósito, pois o WinEdit não fornece notificações de erros no código nem sugere soluções. Como existia uma pergunta que pedia para o participante completar uma linha em branco do código, não queríamos que ele tivesse ajuda do editor.

Validade de Construto: A clareza das perguntas apresentadas aos participantes poderia representar uma ameaça, uma vez que diferentes participantes poderiam interpretar uma mesma pergunta de maneiras distintas. Por isso, avaliamos e ajustamos as perguntas com base no estudo piloto. A forma como medimos o esforço de compreensão poderia ser outra ameaça, pois se trata de algo subjetivo. Para isso, utilizamos procedimentos indicados pela literatura e já utilizadas em outros estudos.

4. Resultados e Discussões

A Figura 1 mostra o tempo médio para analisar e responder as perguntas relativas a cada classe. O tempo médio foi calculado somando-se o tempo que os participantes levaram para responder as quatro perguntas de cada classe e dividindo pelo número de participantes. Podemos observar que a classe Person2 foi a que demandou mais tempo de análise: 22 minutos. Essa classe possui baixa coesão estrutural (LCOM5 = 0,93) e alta coesão conceitual (LCbC = 1). É importante destacar que, apesar de termos selecionados classes com tamanhos similares, não foi possível encontrar classes com exatamente o mesmo número de linhas de código. Porém, Person2 não é a maior classe. Ela tem LOC = 254, enquanto BibliotecaUI2 tem LOC = 274 e é a maior classe.

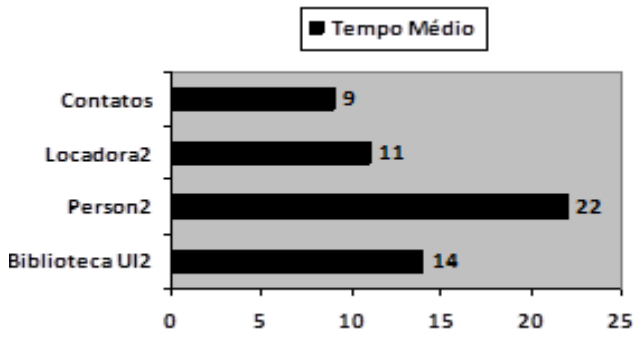

Figura 1. Tempo médio para analisar cada classe, em minutos.

BibliotecaUI2 demandou o segundo maior tempo para ser analisada: 14 minutos. Mesmo assim esse tempo não é muito maior que os tempos de análise requeridos pelas demais classes: Locadora2 (11 minutos) e Contatos (9 minutos). Em termos de coesão, BibliotecaUI2 apresenta características inversas a Person2: alta coesão estrutural (LCOM5 = 0) e baixa coesão conceitual (LCbC = 4).

A classe Contatos é a mais bem coesa tanto conceitualmente (LCbC = 1) quanto estruturalmente (LCOM5 = 0). Por outro lado, a classe Locadora2 apresenta baixa coesão conceitual (LCbC = 4) e baixa coesão estrutural (LCOM5 = 0,6). Apesar do contraste em relação aos valores de coesão, as duas classes requereram os dois menores tempo de análise: 9 minutos para Contatos e 11 minutos para Locadora2.

A Figura 2 mostra o número médio de erros cometidos pelos participantes durante a análise de cada classe. Um erro significa que o participante forneceu resposta errada para uma das questões relacionadas à classe. O número médio de erros por classe é a soma do número de erros para a classe dividido pelo número de participantes.

Os resultados observados na Figura 2 são bem semelhantes aos resultados do tempo médio (Figura 1). Considerando tanto o tempo médio quanto o número médio de erros, a classe Person2 foi a classe que demandou maior esforço para ser compreendida enquanto a classe Contatos demandou o menor esforço. Isso aumenta a confiança do estudo, pois demonstra que não houve casos em que o participante cometeu menos erros em uma classe porque gastou mais tempo analisando-a.

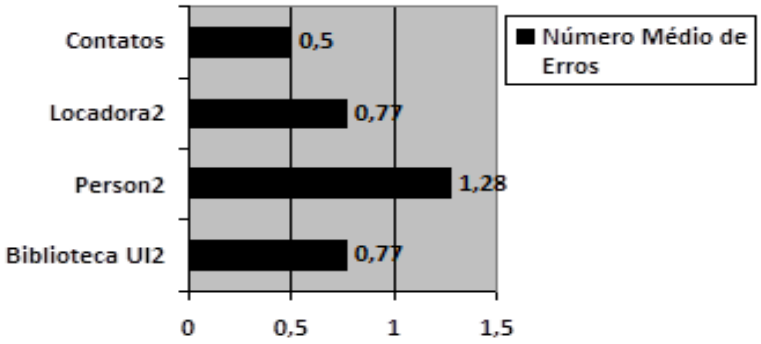

Figura 2. Número médio de erros de cada classe.

Analisando dados sob a ótica do perfil dos participantes, observamos que os sujeitos considerados experientes tiveram melhor desempenho. Para esse grupo o maior número de erros obtidos foi quatro e o maior tempo de análise foi de 68 minutos. Para o grupo considerado pouco experiente o maior número de erros foi oito e o maior tempo gasto foi de 81 minutos.

Ao final do experimento, um questionário de feedback pedia ao participante para indicar a classe que ele achou mais difícil de compreender. As respostas a essa questão confirmaram os dados quantitativos: todos elegeram a classe Person2 como a classe mais difícil de compreender. O principal fator apontado pelos participantes como algo que dificultou o entendimento das classes foi o fato da classe analisada usar outras classes. Para eles, quanto mais dependente de outras classes mais difícil foi para entender a classe. É importante destacar aqui que os participantes não tiveram acesso ao código fonte das outras classes dos sistemas. Essa informação nos levou a coletar para cada uma das classes o valor de uma métrica de acoplamento. O objetivo foi saber até que ponto, as classes analisadas dependiam de outras classes do sistema. Usamos a bem conhecida

métrica Coupling Between Objects – CBO, definida por Chidamber & Kemerer [Chidamber & Kemerer 1994]. Os valores dessa métrica podem ser visualizados na Tabela 2.

Tabela 2. Valor do acoplamento das classes

Métrica	BibliotecaUI2	Person2	Locadora2	Contatos
CBO	3	5	3	1

A classe Person2 tem o maior grau de acoplamento (CBO = 5) dentre as quatro classes utilizadas: ela depende de cinco outras classes do sistema. Provavelmente tenha sido esse o motivo de sua compreensão ter sido considerada a mais difícil, tanto pelos resultados quantitativos, quanto pela opinião dos participantes. Esse resultado nos levou a planejar estudos para avaliar a relação entre acoplamento e esforço de compreensão como trabalhos futuros.

Diante dos resultados, discutimos as questões de pesquisa da seguinte forma:

***Questão 1*: A coesão de uma classe apresenta alguma influência sobre a compreensão do seu código?** Os resultados do estudo não foram suficientes para responder a essa questão. A classe que se mostrou mais difícil de entender (Person2) é a que tem coesão estrutural mais baixa. No entanto, parece que a dificuldade para compreender seu código deveu-se mais ao fato dela ser a classe que mais depende de outras classes do sistema. A classe mais fácil de entender (Contatos) foi a que apresentou coesão, tanto conceitual quanto estrutural, mais alta. No entanto, também é a classe com o menor grau de acoplamento.

***Questão 2*: Há diferenças no esforço de compreensão de classes com diferentes valores de coesão conceitual e coesão estrutural?** Os resultados do estudo também não foram suficientes para responder a essa questão. Se tomarmos com base as classes Person2 e Contatos, poderíamos dizer que coesão estrutural tem relação com compreensão. Porém, temos de novo o fator do alto acoplamento da classe Person2 influenciando. Por outro lado, as classes BibliotecaUI2 e Locadora2 tem: (i) aproximadamente o mesmo nível de esforço de compreensão, (ii) o mesmo grau de coesão conceitual (valores iguais de LCbC) e (iii) o mesmo valor de acoplamento. Como elas apresentam diferentes graus de coesão estrutural (LCOM5 = 0 para BibliotecaUI2 e LCOM5 = 0.6 para Locadora2) e demandam esforço similar para serem compreendidas, podemos dizer que, nesse caso, não houve relação entre coesão estrutural e esforço de compreensão.

5. Conclusão

O estudo preliminar desenvolvido teve objetivo de analisar em que nível o grau de coesão de classes está relacionado ao esforço para compreender o código fonte dessas classes. Os resultados obtidos não permitiram responder as questões de pesquisa, principalmente por não termos anulado um fator de confusão importante: o grau de acoplamento das classes. Um dos pontos positivos do estudo foi o fato de termos tido sucesso em medir o esforço de compreensão de classes.

Diante do contexto, pretendemos realizar outro experimento com um número maior de participantes e classes. Pretendemos tomar mais cuidado para minimizar os fatores de confusão, principalmente os valores de acoplamento das classes. Além disso,

esse estudo nos motivou também a realizar outro estudo para avaliar a relação entre o grau de acoplamento e o esforço de compreensão.

Agradecimentos. Esse trabalho foi apoiado pelo CNPq: Instituto Nacional de Ciência e Tecnologia para Engenharia de Software (processo 573964/2008–4) e Projeto Universal (processo 486662/2013-6)

Referências

Bois, B. D., Demeyer, S.; Verelst, J., Mens, T., Temmerman, M. (2006) "Does God Class Decomposition Affect Comprehensibility?". *International Conference on Software Engineering – IASTED*.

Briand, L. C., Daly, J. W. and Wust, J. K. (1998). "A Unified Framework for Cohesion Measurement in Object-Oriented Systems". *Empirical Software Engineering - An International Journal*, 3(1), pp. 65-117.

Chidamber, S.; Kemerer, C. (1994) "A Metric Suite for Object Oriented Design". *IEEE Transactions on Software Engineering*, 20(6), pp. 476-493.

Dunsmore, A.; Roper, M. (2000) "A Comparative Evaluation of Program Comprehension Measures". *EFoCS SEG Technical Reports*.

Feigenspan, J.; Apel, S.; Liebig, J.; Kastner, C. (2011) "Exploring Software Measures to Assess Program Comprehension". *IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*.

Ferreira, K. A. M.; Bigonha, M. A. S.; Bigonha, R. S.; Almeida, H. C.; Neves, R. C. (2011) "Métrica de Coesão de Responsabilidade – A Utilidade de Métricas de Coesão na Identificação de Classes com Problemas Estruturais". *Simpósio Brasileiro de Qualidade de Software (SBQS)*. Curitiba - Paraná.

Henderson-Sellers, B., Constantine, L. and Graham, M. (1996) "Coupling and Cohesion (Towards a valid metrics suite for object-oriented analysis and design)". *Object Oriented Systems*, vol 3, pp. 143-158.

Liu, Y. et al. "Modeling class cohesion as mixtures of latent topics". (2009) Int'l Conf. on Software Maintenance (ICSM2009), September, Edmonton, p. 233–242.

Marcus, A. & Poshyvanyk, D. (2005) "The Conceptual Cohesion of Classes". Intl' *Conference on Software Maintenance (ICSM '05)*. Washington, DC, pp. 133-142.

Mondal, M.; Rahman, Md. S.; Saha, R. K.; Roy, C. K.; Krinke, J.; Schneider, K. A. (2011) "An Empirical Study of the Impacts of Clones in Software Maintenance". *IEEE 19th International Conference on Program Comprehension (ICPC)*.

Pfleeger, S.; Atlee, J. (2010) "Software Engineering: theory and practice", 4th ed, Prentice Hall.

Silva, B., Sant'Anna, C., Chavez, C. and Garcia, A. (2012) "Concern-Based Cohesion: Unveiling a Hidden Dimension of Cohesion Measurement". *IEEE International Conference on Program Comprehension (ICPC 2012)*, Passau, pp. 103-112.

Silva, B., Sant'Anna, C., Chavez, C. (2014) "An Empirical Study on How Developers Reason about Module Cohesion." *Int'l Conference on Modularity (Modularity 2014)*, Lugano, pp. 121-132.

Rugaber, S. (1995) "Program Comprehension". Working Report. Georgia Institute of Technology. May 1995.