TimeLine Matrix: an on demand view for software evolution analysis

Renato L. Novais[1,2], Paulo R. M. Simões Júnior[1], Manoel Mendonça[1]

[1]Computer Science Department, Federal University of Bahia (UFBA), Bahia, Brazil

[2] Information Technology Department, Federal Institute of Bahia, Campus Santo Amaro, Bahia, Brazil

{renatoln, pauloroberto, mgmendonca}@dcc.ufba.br

Abstract. *Software evolution is one of the most important topics of modern research in software engineering. Many works of software visualization have been proposed to assist in data analysis of software evolution. These studies usually overview the history or show a snapshot of the evolution. Many of them provide details on demand, but only through tooltips or accessing the source code. We present the vision Timeline Matrix, which operates on demand from other views. Once selected the element of interest, this view shows the complete evolution of this element. Allowing to easily navigating in the history, and making comparisons among up to three elements. An example usage of this view in a real scenario is presented, showing how it can help solve evolution comprehension tasks.*

Resumo. *Evolução de software é um dos tópicos mais importantes da pesquisa moderna em engenharia de software. Muitos trabalhos de visualização de software têm sido propostos para ajudar na análise dos dados de evolução. Estes trabalhos, normalmente, apresentam uma visão global ou um retrato instantâneo da evolução do software. Muitos deles oferecem detalhes sob demanda, porém apenas através de tooltips ou acesso ao código fonte. Neste trabalho, apresentamos a visão TimeLine Matrix, a qual funciona sob demanda de outras visões. Uma vez selecionado o elemento de interesse, esta visão mostra a evolução completa deste elemento. Ela permite assim navegar facilmente na história, e fazer comparações entre até três elementos. Um exemplo de uso dessa visão em um cenário real é apresentado, mostrando como ela pode ajudar a resolver tarefas de compreensão de evolução de software.*

1. Introdução

Evolução de software é um dos tópicos mais importantes da pesquisa moderna em engenharia de software. Esta atividade requer a análise de grande quantidade de dados que descrevem sua estrutura atual, bem como sua história. As áreas de métricas e visualização de software têm sido utilizadas para facilitar a compreensão deste cenário. As métricas ajudam a medir e controlar os artefatos que estão evoluindo [Wohlin et al. 2000]. A visualização ajuda a sumarizar os dados complexos em compreensíveis cenários visuais [Diehl 2007][Storey 2005].

As duas áreas são comumente utilizadas em conjunto. Os números apresentados apenas através dos seus valores não são intuitivos para os usuários. A visualização faz uso de metáforas visuais que facilitam a interpretação dos valores das métricas.

Muitos trabalhos de visualização de software têm sido propostos para ajudar na análise de dados de evolução [Eick et al. 1992] [Lanza 2001] [D'Ambros et al. 2009] [Cepeda et al. 2010][Bergel et al. 2011]. A grande maioria mostra os dados através de: i) visões globais (*overview*), que apresentam, na mesma cena visual, toda a história do software, e, ii) visões instantâneas (*snapshots*), que mostram o software em algum momento da história. Estes trabalhos seguem o mantra de visualização de Shneiderman (1996): *Overview first, details on demand*. Entretanto, a parte de "detalhes sob demanda" destes trabalhos são, normalmente, interagir com as visões para acessar mais propriedades do módulo de software na versão de interesse, ou até mesmo acessar o código fonte dessa versão.

Nos últimos três anos, temos trabalhando num ambiente de visualização de evolução de software chamado *SourceMiner Evolution* – SME[1] – [Novais et al. 2011a][Novais et al. 2011b][Novais et al. 2012]. Esse ambiente permite visualizar a evolução de software seguindo diferentes Estratégias de Análise[2] de evolução de software através recursos visuais. Até esse momento, estão disponíveis a estratégia diferencial relativa [Novais et al. 2011a], estratégia diferencial absoluta [Novais et al. 2012], estratégia temporal overview [Novais et al. 2011b]. Apesar da gama de recursos disponíveis (e.g. detalhes sob demanda, da forma como citado anteriormente), percebemos, nos diversos estudos realizados, a necessidade de uma visão que fornecesse detalhes sob demanda de uma forma diferente das atualmente propostas na literatura. A ideia é poder selecionar um elemento de software de interesse nas outras visões, e poder visualizar a sua evolução separadamente, vendo todo o histórico do elemento de interesse.

Nesse contexto, desenvolvemos uma nova visão: *TimeLine Matrix* (TLM). Esta visão é altamente configurável, permitindo visualizar a evolução de até três elementos de software por vez. Os atributos visuais podem ser facilmente mapeados aos atributos reais disponíveis no ambiente (e.g. métricas de software, mapeamento de funcionalidades (*features*) no código fonte). Desta forma, é possível fazer a análise da evolução do software por elemento, podendo inclusive fazer a comparação da evolução de até três elementos de software diferentes.

Além dessa introdução, este artigo está organizado como se segue. Na Seção 2, serão apresentados alguns trabalhos relacionados. Na Seção 3, será apresentada essa nova visualização desenvolvida, mostrando todos os recursos disponíveis. Exemplo de uso, baseado em um dos nossos estudos anteriores, é apresentado na Seção 4. Por fim, a Seção 5 conclui este artigo.

---

[1] Website do SME: http://www.sourceminer.org/

[2] Uma estratégia de análise define como a evolução de um artefato de software é apresentada para análise. A representação dessa evolução pode ser representada por diferentes maneiras que facilitam (ou dificultam) a compreensão dos fenômenos associados às mudanças. Não é objetivo desse trabalho explicar estas estratégias. Para um melhor entendimento sugerimos nosso trabalhos anteriores, cujas referências estão disponíveis no texto.

2. Trabalhos Relacionados

A visualização tem sido bastante utilizada para análise de evolução de software. Ao longo do anos, muitos trabalhos têm sido propostos. Trabalhos com diferentes visões, estratégias de análise, perspectivas, e focando em tarefas diferentes de Engenharia de Software.

Visualização de evolução de software não é algo recente. Um dos primeiros trabalhos é datado de 1992, por Eick et al. O *SeeSoft*, proposto por estes autores, mapeia cada linha de código a linhas formadas por pixels. Dentre as suas funcionalidades, estão desde a compreensão de uma versão, quanto a visualização de mudanças recentes no código fonte. Outro trabalho que tem uma importância histórica bastante relevante, é o *Evolution Matrix* [Lanza 2001]. Neste trabalho, o autor propõe uma matriz de evolução mostrando todas as classes do sistemas em todas as versões disponíveis.

Mais recentemente, muitos trabalhos foram publicados com diferentes propósitos. D'Ambros et al. (2009) propuseram uma visualização em radar para analisar acoplamento lógico entre os módulos do software. Cepeda et al. (2010) propuseram uma visualização que permite detectar e externalizar a evolução do design do software. Em [Bergel et al. 2011] é apresentada outra visualização que permite comparar perfis de diferentes versões do software e destacar mudanças críticas de desempenho no sistema.

A *TimeLine Matrix* segue a mesma linha da *Polymetric Views* de Lanza e Ducasse (2003), permitindo o mapeamento de atributos reais a atributos visuais. Do ponto de vista de paradigma visual ela se assemelha ao *Evolution Matrix*. Porém se difere dela e dos outros aqui apresentados, por ser uma visão sob demanda, integrada a outras visões, e facilmente configurável. Assim, o usuário pode configurar a visualização de acordo com seu objetivo, e da forma que mais facilite a sua compreensão da evolução do software.

3. TimeLine Matrix

A *TimeLine Matrix* (TLM) é uma nova visão que permite a visualização da evolução de elementos de software sob demanda. A Figura 1 apresenta uma visão geral do SME dando ênfase a visão TLM. O SME possui três visões gerais (*Treemap*, *Polymetric*, *Coupling*) que utilizam as estratégias de análise Diferencial Relativa e Diferencial Absoluta. A quarta visão é a Coordenadas paralelas. Ela endereça a estratégia Temporal overview, e também funciona sob demanda a partir das três primeiras visões. A partir de qualquer uma das três visões gerais é possível selecionar um elemento de software de interesse, em qualquer versão disponível e solicitar detalhes sob demanda na visão TLM.

A visão TLM foi projetada no formato de uma matriz 3x3. Cada linha desta matriz é chamada de *timeline* de evolução. Cada coluna representa uma versão do elemento de software em análise. Cada *timeline* mostra por vez até três versões do elemento. É possível selecionar para cada *timeline* um elemento de software diferente ou o mesmo elemento de software para todos os *timelines*. Neste último caso, será possível ver até nove versões do mesmo elemento na mesma cena visual. Recursos de navegação entre as versões estão disponíveis na visão.

Um dos recursos mais importante dessa visão, o qual dá a ela uma característica bastante flexível, é a possibilidade de mapear facilmente atributos reais (e.g. métricas de software) a três atributos visuais disponíveis nessa visão (tonalidade da cor, largura e altura do retângulo). Desta forma, o usuário pode selecionar quaisquer métricas, dentre as disponíveis, e visualizar a evolução de acordo com seu interesse. O crescimento do atributo real é mapeado na coloração mais escura do elemento visual, bem como na maior largura ou altura do elemento visual. O decrescimento é tratado de forma oposta. Três cores distintas são utilizadas nessa visão: azul, vermelho, e verde para representar pacotes, classes e métodos, respectivamente.

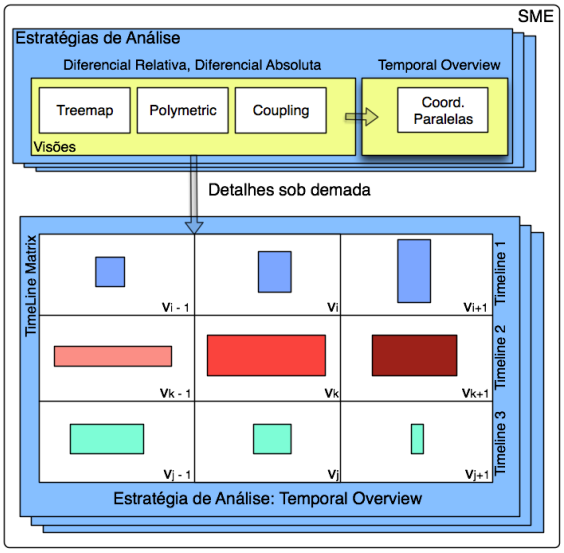

Figura 1. Uma visão geral do SME com ênfase na *TimeLine Matrix*

No *Timeline* 1 da Figura 1, por exemplo, é possível observar que houve um crescimento na métrica mapeada ao atributo visual "altura do retângulo" da versão v_{i-1} até v_{i+1}. No *Timeline* 2 é possível observar que a "tonalidade" do vermelho está escurecendo da versão v_{k-1} até v_{k+1}, de onde conclui-se que a métrica mapeada também está crescendo. Por fim, no *Timeline* 3 há um decrescimento da métrica associada a "largura do retângulo" da versão v_{j-1} até v_{j+1}.

Através dessa nova visão, o usuário pode comparar a evolução de até três elementos diferentes de software. Isso permite análises comparativas e percepção de tendências entre os elementos.

Diversas métricas estão disponíveis para uso na TLM. Como exemplos, podem ser citadas Linha de Código, Número de Métodos, complexidade ciclomática, acoplamento aferente e eferente, Número de classes, etc. Cada atributo de software possui seu conjunto de métricas específico, de tal forma que não é possível mapear atributos reais a atributos visuais de forma inconsistente (e.g. mapear na "largura do retângulo" o número de métodos, quando o elemento sendo visualizado é o próprio método).

A Figura 2 apresenta a visão TLM em ação. É possível observar as nove células da matriz, sendo que sob cada célula, é apresentado a versão do elemento. Botões de navegação *forward* e *backward* existem em cada *timeline*. Por fim, sobre cada *timeline* é

apresentado o nome do elemento sendo visualizado. No lado esquerdo da Figura 2 é apresentado a visão *TimeLine Filter*, que permite fazer os mapeamentos dos atributos reais aos atributos visuais para cada um dos três *timelines*.

Essa visão possui também a possibilidade de se obter mais detalhes sob demanda. Através do *tooltip*, o usuário pode acessar informações detalhadas do elemento sendo visualizado, como por exemplo as funcionalidades (*features*) sendo realizada pelo elemento de software. Visualização de evolução de *features* é um recurso disponível no SME [Novais et al. 2012]. É possível ainda acessar o código fonte a partir dos elementos visuais. Outra funcionalidade é a possibilidade de se fazer o caminho inverso entre as visões. A partir de um elemento (pacote, classe ou método) na TLM, pode ser solicitado, através de menu *pop up*, para que o elemento seja visualizado na visão *Treemap*. Esta operação abre, por exemplo, uma classe em detalhe na *Treemap*, mostrando todos os seus métodos.

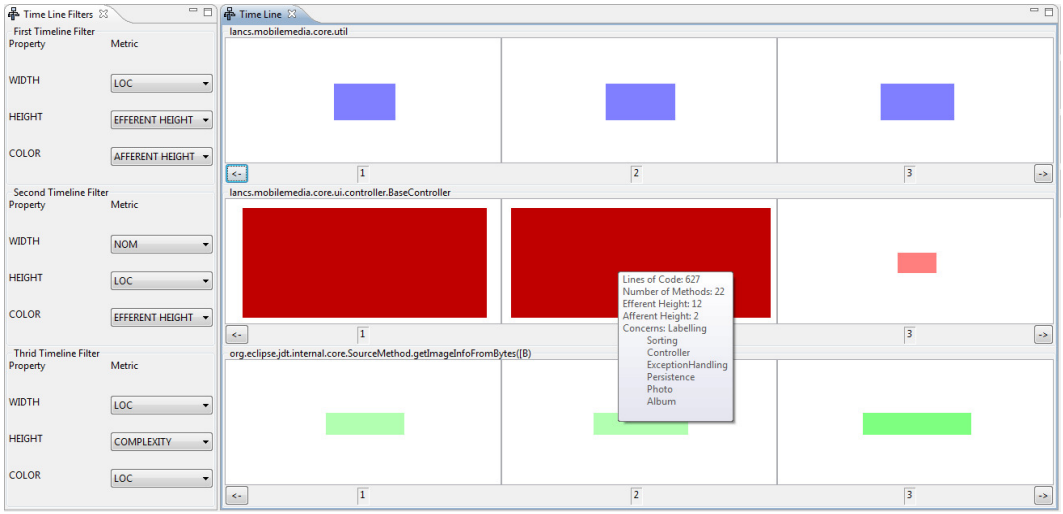

Figura 2 – Visão *TimeLime Matrix* em ação

4. Exemplo de Uso

Nesta seção será apresentado um exemplo de uso da TLM. Esta nova visão surgiu a partir dos estudos realizados anteriormente, principalmente a partir do experimento controlado descrito em [Novais et al. 2012]. Nesse experimento, foi feita uma comparação entre o SME e *ConcernMapper* [Robillard e Weigand-Warr 2005] em relação a análise de evolução de *features*. O estudo contou com a participação de 20 desenvolvedores, sendo 10 de Salvador – Bahia e 10 do Rio de Janeiro – RJ. Os participantes deveriam realizar um conjunto de seis atividades, relacionadas à compreensão de evolução de *features*, em cinco versões de um software industrial. As tarefas estavam classificadas em três objetivos diferentes: Análise de Evolução de *Feature*, Análise de Entrelaçamento de *Feature*, e Análise de Dependência de *Features*.

Duas hipóteses foram definidas para o estudo: H1) O SME decresce o tempo gasto nas tarefas de compreensão de evolução de *features*; H2) O SME melhora a corretude das tarefas de compreensão de evolução de *features*. Após a análise estatística dos dados, pôde-se observar que o SME decresceu o tempo gasto, em média, em 8.95%

em relação ao *ConcernMapper*, entretanto não foi possível rejeitar a hipótese nula, consequentemente não foi possível aceitar a hipótese H1. Por outro lado, em relação a hipótese H2, o SME aumentou a corretude, em média, em 26.94% quando comparado com o *ConcernMapper*. Os resultados, neste caso, foram relevantes, podendo inclusive aceitar a hipótese (H2) de que o SME melhora a corretude.

Diversos fatores relacionado aos tratamentos, ao objeto de estudo, e *design* do experimento foram discutidos a fim de entender os resultados encontrados. Foi observado, por exemplo, que o SME não obteve melhores resultados em relação a eficiência, devido ao tempo gasto para a representação gráfica do software sob estudo, nas visões utilizadas. Como o software era grande, as visões levavam tempo para serem redesenhadas a cada mudança de versão. Isto ficou evidente principalmente em relação a duas tarefas específicas. Essas tarefas são discutidas a seguir, mostrando como os participantes utilizaram o SME durante o experimento, e como seria, caso a visão *TimeLine Matrix* estivesse disponível durante esse experimento. As duas tarefas são relacionadas à análise de entrelaçamento de *features*.

Tarefa T4: Quais são as features realizadas pela classe DBConnection em cada versão?

Para realizar essa tarefa durante o experimento, os participantes tiveram basicamente que realizar um conjunto de ações sobre o SME: i) selecionar a versão 1 para ser visualizada; ii) utilizar o filtro para encontrar a classe *DBConnection* na visão; iii) requisitar detalhes sob demanda (e.g. *tooltip*) para descobrir quais as *features* realizadas por esse elemento de software. Todos esses passos teriam que ser repetidos para cada uma das outras quatro versões seguintes. O grande problema enfrentado foi no momento de redesenhar as visões quando havia a mudança de versão. Isso aconteceu principalmente pelo fato do objeto de estudo ser um sistema de larga escala, com, em média, 500 classes em cada versão.

Em atividades desse tipo, é possível observar que, uma vez encontrado o elemento de software de interesse, não se faz mais necessário visualizar os outros elementos de software. Isto foi melhorado com a implementação da TLM. Utilizando a TLM, o participante teria que realizar os seguintes passos: i) selecionar a versão 1 para ser visualizada; ii) utilizar o filtro para encontrar a classe *DBConnection* na visão; iii) requisitar para que o elemento de interesse fosse apresentado em algum *timeline* da TLM; iv) requisitar detalhes sob demanda na visão TLM, recuperando as *features* para cada versão da classe em questão. O usuário poderia visualizar as cinco versões de uma só vez, utilizando para isso dois *timelines* distintos, ou navegar em um único *timeline* através dos componentes de navegação disponíveis. Neste caso, não existe mais o tempo gasto para o processamento e representação de todos os elementos de cada uma das versões do software sob estudo.

Tarefa T5: A classe ServerConfig está realizando mais de uma feature ao longo da evolução? Se sim, quais são as features? Para cada feature, em cada versão, quais são os métodos realizando ela?

O processo de realização dessa tarefa é bem semelhante ao da tarefa anterior. Para realizá-la durante o experimento, os participantes tiveram basicamente que realizar o seguinte conjunto de ações sobre o SME: i) selecionar a versão 1 para ser visualizada; ii) utilizar o filtro para encontrar a classe *ServerConfig* na visão; iii) requisitar detalhes

sob demanda (e.g. *tooltip*) para descobrir quais as *features* realizadas por esse elemento de software; iv) identificar, para cada método dessa classe, quais *features* eles realizam. Da mesma forma, todos esses passos teriam que ser repetidos para cada uma das outras quatro versões seguintes. O problema de redesenhar as visões quando havia a mudança de versão também impactou no resultado final do experimento.

Essa atividade também seria mais facilmente realizada com a utilização da TLM. Com esta visão, o participante teria que realizar basicamente os seguintes passos da tarefa anterior: i) selecionar a versão 1 para ser visualizada; ii) utilizar o filtro para encontrar a classe *ServerConfig* na visão; iii) requisitar para que o elemento de interesse fosse apresentado em algum *timeline* da TLM; iv) requisitar detalhes sob demanda na visão TLM, recuperando as *features* para cada versão da classe em questão. A principal diferença aqui, é que neste caso é necessário identificar as *features* por método, e o TLM estaria apresentando a classe. Para resolver esse problema, o SME dispõe de um recurso de integração entre todas as visões. Neste caso, deve-se solicitar para que a classe de interesse seja visualizada na visão *Treemap*. Assim, as *features* por métodos seriam coletadas na visão *Treemap*, enquanto que a navegação entre as versões seria realizada através do TLM. Desta forma, o uso do TML permitiria novamente a diminuição do tempo gasto com o redesenhar das visões.

5. Conclusão

Dada a importância da evolução de software, diversos trabalhos de visualização de evolução de software têm sido propostos. Estes trabalhos oferecem ricas visões, com possibilidade de acesso a detalhes sob demanda, através de *tooltips* ou acesso ao código fonte. Entretanto, algumas tarefas de compreensão de evolução de software requerem um outro tipo de detalhes sob demanda: uma vez selecionado um elemento de interesse numa visão global, poder visualizar a evolução apenas desse elemento.

No contexto desse trabalho, foi apresentado a visão *TimeLine Matrix*. Uma visão sob demanda que está integrada ao ambiente de visualização de evolução de software *SourceMiner Evolution*. Esta nova visão permite que o usuário navegue na história de um elemento de software (pacote, classe ou método) de forma rápida e prática. Mapeamentos de recursos reais a recursos visuais são disponíveis na visão, tornando-a bastante flexível. Desta forma, o usuário pode configurá-la de acordo com o sua tarefa de compreensão de evolução.

Para mostrar sua aplicabilidade, foi apresentado como a *TimeLine Matrix* ajudaria a resolver duas tarefas de compreensão de evolução de *features*, extraídas de um estudo controlado realizado anteriormente. Entretanto, é sabido que sua efetividade ainda precisa ser avaliada. Isto está sendo planejado, por exemplo, através da replicação do estudo apresentado. Espera-se que os usuários do SME se beneficiem com as funcionalidades disponibilizadas através da nova visão *TimeLine Matrix*.

Referências

Bergel, A., Bañados, F., Robbes, R. and Binder, W. (2011), “Execution profiling blueprints” In Softw: Pract. Exper.

Cepeda, R. D. S. V., Magdaleno, A. M., Murta, L. G. P., Werner, C. M. L. (2010). “EvolTrack: Improving Design Evolution Awareness in Software Development”, In

Journal of the Brazilian Computer Society (JBCS), Volume 16, Number 2 (2010), 117-131.

D'Ambros, M., Lanza, M. and Lungu, M. (2009). “Visualizing Co-Change Information with the Evolution Radar”, In IEEE Trans. Softw. Eng. 35, 5 (September 2009), 720-735.

Diehl, S. (2007). “Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software”, Springer-Verlag New York, Inc.

Eick, S. G., Steffen, J. L., and Sumner, E. E. Jr. (1992). “Seesoft-A Tool for Visualizing Line Oriented Software Statistics”, In IEEE Trans. Softw. Eng. 18, 11 (November 1992), 957-968.

Lanza, M. (2001). “The evolution matrix: recovering software evolution using software visualization techniques”, In Proceedings of the 4th International Workshop on Principles of Software Evolution (IWPSE '01). ACM, New York, NY, USA, 37-42.

Lanza, M. and Ducasse, S. (2003). Polymetric Views-A Lightweight Visual Approach to Reverse Engineering. IEEE Trans. Softw. Eng. 29, 9, 782-795.

Novais, R. L. ; Lima, C. A. N. ; Carneiro, G. F. ; Simoes Junior, P. R. M. ; Mendonça Neto, M. G. (2011b). “An Interactive Differential and Temporal Approach to Visually Analyze Software Evolution”, In: 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis, Williamsburg. VISSOFT11, v. 1. p. 88-91.

Novais, R. L., Carneiro, G. F., Simoes Junior, P. R. M., Mendonça Neto, M. G. (2011a). “On the Use of Software Visualization to Analyze Software Evolution - An Interactive Differential Approach”, In: 13th International Conference on Enterprise Information System, Beijing. v. 3. p. 15-24.

Novais, R., Nunes, C., Lima, C., Cirilo, E., Dantas, F., Garcia, A., Mendonça, M. (2012). “On the Proactive and Interactive Visualization for Feature Evolution Comprehension: An Industrial Investigation”, Proceedings of ICSE, SE in Practice. Zurich, Switzerland. IEEE, pp 1044-1053.

Robillard, M., Weigand-Warr, F. (2005). “Concernmapper: simple view- based separation of scattered concerns”, Proc. of the OOPSLA workshop on Eclipse Technology, ACM, pp. 65–69.

Shneiderman, B. (1996). “The eyes have it: A task by data type taxonomy for information visualizations”, In VL ’96: Proceedings of the 1996 IEEE Symposium on Visual Languages, page 336,Washington, DC, USA. IEEE Computer Society.

Storey, M. D., Čubranić, D., and German, D. M. (2005). “On the use of visualization to support awareness of human activities in software development: a survey and a framework”, In Proceedings of the 2005 ACM Symposium on Software Visualization (St. Louis, Missouri, May 14 - 15, 2005). SoftVis '05. ACM, New York, NY, 193-202.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A. (2000). “Experimentation in Software Engineering: An Introduction”, Kluwer Academic Publishers, Norwell, MA, USA.