Avaliação dos Efeitos da Utilização do TDD na Qualidade Interna do Produto

Carlos H. Zacarias[1], Josué B. Santos[1], Reinaldo Cabral[2], Ana Regina Rocha[2]

[1]Departamento de Tecnologia da Informação
Secretaria Executiva de Fazenda do Estado de Alagoas (SEFAZ-AL)
CEP 57017-900 – Maceió – AL– Brasil

[2] Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ
Caixa Postal 68.511 – CEP 21945-970 – Rio de Janeiro – RJ – Brasil

{josuesantos,carlossantos}@sefaz.al.gov.br,
{cabral,darocha}@cos.ufrj.br

Abstract. *The search for efficiency and effectiveness in software development and maintenance is generally supported by the adoption of methods and technologies that may have a great impact in the organization. To prevent the organizations from unexpected results it is important to evaluate these technologies before their institutionalization. This paper presents the experience of the introduction of TDD at SEFAZ-AL and the results achieved.*

Resumo. *A busca por mais eficiência e eficácia no desenvolvimento e manutenção de software geralmente é subsidiada pela adoção de tecnologias que podem causar grande impacto na forma como a organização trabalha. Para evitar resultados inesperados é imprescindível avaliar métodos e tecnologias antes de institucionalizá-las nas organizações. Este trabalho relata a introdução do TDD na SEFAZ-AL e os resultados obtidos com a experiência.*

1. Introdução

Com a disseminação dos métodos ágeis, o TDD (Test Driven Development – Desenvolvimento Orientado a Testes) tem ganhado visibilidade e tem sido alvo de diversos estudos que procuram avaliar sua efetividade, especialmente com relação à produtividade dos programadores e à redução do número de defeitos [George e Williams 2003] [Geras et al. 2004] [Müller e Hagner 2002] [Williams et al. 2003]. Dados apresentados por Williams et al. (2003) apontam uma redução de 40% no número de defeitos capturados durante as atividades de verificação e testes de regressão, sem impactar na produtividade da equipe. Em contrapartida, o estudo conduzido por Müller e Hagner (2002) indicou que o benefício em utilizar a técnica é mínimo ou inexiste ao analisar aspectos relacionados à produtividade e confiabilidade.

Devido à ausência de evidências que comprovem a efetividade do TDD e ciente do impacto potencial nas atividades cotidianas da organização, a Diretoria de Tecnologia da Informação (DTI) da Secretaria Executiva de Fazenda do Estado de Alagoas (SEFAZ-AL) decidiu realizar uma avaliação antes de institucionalizar a sua utilização nas atividades de desenvolvimento e manutenção de software.

Além desta breve introdução, o trabalho possui mais quatro seções: a segunda seção caracteriza sucintamente o TDD, a terceira seção descreve o contexto ao qual a tecnologia está sendo inserida, a seção quatro discorre sobre o estudo realizado e, por fim, uma breve conclusão é apresentada na seção.

2. Desenvolvimento Orientado a Testes (TDD)

O TDD surgiu conjuntamente com a ascensão dos métodos ágeis, ambos possuindo raízes nos modelos de processo iterativo, incremental e evolucionário. Há indícios que a idéia tenha sido posta em prática há décadas [Janzen e Saiedian 2005]. Um bom exemplo é o Cleanroom [Mills et al. 1987], processo de engenharia de software introduzido nos anos 80 que utiliza verificação formal do design no início do processo de desenvolvimento.

A primeira tentativa clara de definir o TDD como uma prática surgiu no XP (eXtreme Programming) [Beck 1999], sendo uma das suas práticas centrais aplicada durante a análise, projeto, especificação e testes [Janzen e Saiedian 2005].

No TDD, os testes que especificam a funcionalidade são escritos antes da própria funcionalidade [Williams et al. 2003]. Os testes guiam o projeto do software em construção e têm a finalidade de especificação, juntamente com às comumente atribuídas, de verificação e validação [Geras et al. 2004]. Cada funcionalidade só é considerada construída quando todos os testes, novos ou pré-existentes, são executados com sucesso. Este fato implica a existência de um entendimento prévio do requisito a ser construído para que seja possível escrever uma especificação que detalha um pequeno aspecto do comportamento de uma forma concisa, não ambígua e executável [Astels, 2007].

Para os que utilizam TDD, uma das grandes vantagens, e uma das prováveis causas do seu sucesso, é a possibilidade de diminuição da distância entre a decisão (*design*), e o *feedback* (resultado da implementação do design). Com TDD é possível diminuir esta distância o quanto se queira, uma vez que o ciclo testar-codificar fornece um *feedback* constante ao programador [Beck 2002], sendo esta a forma adotada nas metodologias ágeis [Beck 1999].

3. A Diretoria de Tecnologia da Informação (DTI) da SEFAZ-AL

Em 1997, a Secretaria Executiva de Fazenda do Estado de Alagoas (SEFAZ-AL), no intuito de ter maior controle sobre as informações fiscais, criou o PIF - Programa de Informatização Fazendária. Os sistemas que antes eram mantidos pelo IPD (Instituto de Processamento de Dados) passaram a ser mantidos por funcionários do próprio fisco. Na ocasião o setor de informática da SEFAZ-AL terceirizou quase todos os recursos humanos em termos de desenvolvimento.

Há cerca de quatro anos a SEFAZ-AL realizou um concurso público para contratação de pessoal especializado na área de tecnologia da informação com o intuito de ter maior controle sobre suas aplicações, depender menos de terceirizações e melhorar a qualidade do serviço realizado, principalmente no que diz respeito à qualidade de software. Aos poucos, o setor de informática começou a ganhar status de departamento e, em 2005, foi transformado em Diretoria de Tecnologia da Informação

(DTI). Hoje a diretoria conta com três gerências: Gerência de Apoio ao Usuário, Gerência de Rede e Infra-estrutura e Gerência de Desenvolvimento, na qual está o foco neste trabalho.

Uma das principais dificuldades da DTI reside na escassez de tempo para a realização de melhorias, uma vez que a maioria das pessoas está assoberbada com a manutenção dos sistemas existentes. Este fato tem sido acentuado devido à ausência de documentação, ausência de padrões de desenvolvimento e ausência de testes confiáveis e repetíveis, o que tem tornado a tarefa de manutenção ainda mais árdua.

O Departamento conta hoje com cerca de 50 profissionais, dos quais 30 são desenvolvedores (na prática, não há distinção muito clara entre analistas e programadores) (vide Tabela 1).

Tabela 1. Distribuição dos profissionais e soluções em produção por tecnologia

	JAVA	FORMS	PHP	DELPHI	PL/SQL	NATURAL	ASP
Profissionais (%)	37,93	34,48	6,9	6,9	3,45	3,45	3,45
Soluções (%)	6,67	33,33	16,6	3,33	6,67	3,33	--

Atualmente a SEFAZ-AL possui 29 sistemas em produção, 79,5% desenvolvidos com o paradigma Estruturado, e três em desenvolvimento utilizando o paradigma O.O. Embora apenas 6,67% das soluções estejam na linguagem Java, 37,93% dos profissionais estão trabalhando com a tecnologia, devido à determinação da alta gerência, que decidiu adotar a tecnologia para todos os sistemas considerados críticos, sejam inéditos ou versões evolutivas.

Embora a utilização de procedimentos comuns na DTI seja visível, não há processos definidos para orientar o desenvolvimento e a manutenção de software. Esta liberdade para execução das atividades tem levado a conseqüências indesejáveis, dentre elas: (i) ausência da especificação dos requisitos não-funcionais, que fica a critério dos desenvolvedores; (ii) alta taxa de inclusão de defeitos, tanto na fase de codificação quanto durante a manutenção; (iii) documentação desatualizada, antes mesmo de iniciar a codificação dos sistemas; (iv) decaimento da qualidade do código, o que dificulta o entendimento e torna as mudanças cada vez mais difíceis de serem realizadas; e (v) baixa cobertura de testes, a maior parte dos sistemas entra em produção com um conjunto reduzido de testes, além de ser comum que o desenvolvedor realize seus próprios testes exploratórios ad-hoc.

Visando buscar um maior nível de qualidade nos softwares produzidos e mantidos pela SEFAZ-AL, foi designada uma equipe para elaborar propostas para a melhoria das atividades realizadas na DTI. Levando em conta as lições aprendidas com experiências anteriores, optou-se por uma estratégia visando à introdução paulatina de mudanças de forma a minimizar possíveis resistências da equipe.

A prioridade era aumentar a manutenibilidade do software desenvolvido e mantido pela DTI. Desta forma, o uso de TDD se mostrou alinhado com as necessidades mais urgentes da DTI.

Em um primeiro momento foi fornecido um conjunto de treinamentos relacionados às tecnologias requeridas (Java, Programação OO, Design Patterns, Automação de Testes, Reestruturação de Código e TDD). Durante estes treinamentos foram apresentados diversos casos de sucesso na aplicação das tecnologias. Apesar disso, ainda havia desconfiança, na equipe, com relação às "novidades" apresentadas. Assim, frente às incertezas associadas à adoção da técnica, a equipe de melhoria propôs a adoção do TDD em caráter experimental, optando pela realização de um estudo, orientado por pesquisadores do Laboratório de Engenharia de Software da COPPE/UFRJ, conforme descrito na próxima seção.

4. Avaliação do TDD no Contexto da SEFAZ/AL

Esta seção apresenta o estudo realizado com o intuito de avaliar a aplicação do TDD na SEFAZ. Para avaliação, foram selecionados quatro sistemas: o Cadsinc (C), desenvolvido com o uso de TDD e mais três que não utilizaram TDD. A pedido da gerência, estes terão seus nomes omitidos e serão chamados de sistemas F, S e E. A tabela 2 provê uma caracterização dos sistemas. Todos os projetos foram desenvolvidos em Java, para plataforma Web e utilizando o paradigma OO.

Tabela 2. Caracterização sistemas utilizados no estudo

	C	F	S	E
Tamanho (Linhas de Código)	22.073	168.088	14.248	18.420
Tamanho da Equipe	9 pessoas	25 pessoas	1 pessoa	1 pessoa

Baseado no método GQM (BASILI et al., 1994) foram definidos para este estudo objetivos, questões e métricas, tal como descrito nas próximas seções.

4.1. Objetivo

Analisar: a aplicação do TDD

Com o propósito de: avaliar os benefícios obtidos com a aplicação do TDD

Com respeito a: estrutura interna do código fonte

Do ponto de vista do: desenvolvedor

No contexto da: DTI- SEFAZ.

4.2. Questões e Métricas

Chidamber e Kemerer (1994) afirmam que a estrutura interna do código fonte tem características que podem indicar a qualidade do software produzido. Esses autores preconizam que a qualidade da estrutura interna do código fonte diz respeito ao software (produto) e não leva em consideração o processo de construção e outros fatores que influenciam na construção do código. São exemplos dessas características: (i) Complexidade, medida através da complexidade ciclomática (CC); (ii) Acoplamento, representada pela medida Coupling Between Objects (CBO); (iii) Coesão, que neste contexto foi avaliada pela medida oposta, ou seja, a falta de coesão (LCOM); e (iv) Distância da Seqüência Principal, representando a reusabilidade (D).

Para atingir o objetivo declarado para o estudo, quatro questões de pesquisa foram definidas (vide Tabela 3).

Tabela 3. Questões e Métricas

Questões	Métricas	Premissas e Expectativas
Q1. A aplicação do TDD melhorará a testabilidade e a confiabilidade?	Complexidade Ciclomática (CC)	A testabilidade e a confiabilidade final podem ser mensuradas através da complexidade ciclomática [Watson e McCabe 1996]. Portanto, espera-se que com o uso do TDD a métrica apresente um valor menor, isto é, que a testabilidade e a confiabilidade do código seja melhor do que de aquele produzido sem a adoção do TDD.
Q2. A aplicação do TDD aumentará a coesão entre as classes?	Falta de coesão (LCOM)	Segundo Chidamber e Kemerer (1994) a falta de coesão aumenta a complexidade além de indicar possíveis erros de design e torna as classes mais difíceis de serem testadas. Espera-se que com o uso do TDD diminua a falta de coesão (aumente a coesão), diminuindo assim a complexidade do código.
Q3. A aplicação do TDD diminuirá o grau de acoplamento dos sistemas?	Acoplamento (CBO)	Chidamber e Kemerer (1994) afirmam que o acoplamento excessivo é ruim para modularidade e diminui o reuso, além de ser um indicativo de quão complexo pode ser testar. Espera-se que com o uso do TDD o reuso seja facilitado e a testabilidade seja ampliada, ou seja, diminua o acoplamento.
Q4. A aplicação do TDD diminuirá a distância da seqüência principal?	Distância da Seqüência Principal (D)	Martin (2003) afirma que quanto mais próximo da Seqüência melhor é a relação entre abstração e estabilidade de um componente, fornecendo assim uma grande ajuda na definição de quais componentes possuem maior reusabilidade. A expectativa é que o uso do TDD auxilie na diminuição da distância, isto é potencialize o reuso.

4.3. Hipóteses

Hipótese Nula (**H0**): A qualidade da estrutura interna do código fonte dos sistemas desenvolvidos **com** TDD (CTDD) é menor do que a daqueles que foram desenvolvidos **sem** TDD (STDD).

$$CTDD < STDD \Rightarrow \begin{array}{l} ((CC_C > CC_F) \vee (CC_C > CC_S) \vee (CC_C > CC_E)) \wedge \\ ((LCOM_C > LCOM_F) \vee (LCOM_C > LCOM_S) \vee (LCOM_C > LCOM_E)) \wedge \\ ((CBO_C > CBO_F) \vee (CBO_C > CBO_S) \vee (CBO_C > CBO_E)) \wedge \\ ((D_C > D_F) \vee (D_C > D_S) \vee (D_C > D_E)) \end{array}$$

Hipótese Alternativa (**H1**): A qualidade da estrutura interna do código fonte dos sistemas desenvolvidos **com** TDD (CTDD) não é menor que a daqueles que foram desenvolvidos **sem** TDD (STDD).

$$CTDD \geq STDD \Rightarrow \begin{array}{l} ((CC_C \leq CC_F) \vee (CC_C \leq CC_S) \vee (CC_C \leq CC_E)) \wedge \\ ((LCOM_C \leq LCOM_F) \vee (LCOM_C \leq LCOM_S) \vee (LCOM_C \leq LCOM_E)) \wedge \\ ((CBO_C \leq CBO_F) \vee (CBO_C \leq CBO_S) \vee (CBO_C \leq CBO_E)) \wedge \\ ((D_C \leq D_F) \vee (D_C \leq D_S) \vee (D_C \leq D_E)) \end{array}$$

4.4. Coleta das Métricas

Para este estudo foram pesquisadas diversas ferramentas proprietárias e de código aberto: JDepend[1] (open source), Borland Together[2] (trial), JavaNCSS[3] (open source),

[1] JDepend: http://clarkware.com/software/JDepend.html

Metrics[4] (open source) e Dependency Finder[5] (open source). Todas as ferramentas open source apresentaram dificuldades para serem instaladas e/ou configuradas. Apenas a ferramenta Together possuía todas as métricas selecionadas para este trabalho, no entanto, por razões desconhecidas, a distância da seqüência principal não estava sendo calculada. Esta ferramenta foi utilizada para obter os dados para as métricas LCOM, CBO e CC. A ferramenta JDepend foi utilizada apenas para o cálculo da métrica D.

Todos os dados brutos coletados foram gerados em arquivo texto e processados em uma planilha eletrônica. As tabelas 4, 5, 6 e 7, apresentam uma síntese dos dados.

Tabela 4. Complexidade Ciclomática (CC)

Sistema	Média	Desvio Padrão	Máximo	Observação
C	**1,757**	1,799	37	Os números obtidos estão dentro da expectativa inicial. O sistema C teve a menor complexidade ciclomática média dentre os sistemas analisados.
E	2,243	2,834	64	
S	1,832	1,784	17	
F	2,665	5,064	224	

Tabela 5. Grau de Acoplamento entre os Objetos (CBO)

Sistema	Média	Desvio Padrão	Máximo	Observação
C	9,528	10,057	76	Os números obtidos foram contrários às expectativas, a média mais alta de acoplamento foi a do projeto C, o único construído com TDD.
S	5,641	6,840	39	
E	**0,992**	2,934	41	
F	8,622	11,435	82	

Tabela 6. Falta de Coesão (LCOM)

Sistema	Média	Desvio Padrão	Máximo	Observação
C	153,249	490,881	4301	Os números obtidos foram contrários às expectativas, a média mais alta da falta de coesão foi a do sistema C, o único construído com TDD.
S	**66,898**	146,230	988	
E	119,247	180,444	1046	
F	123,280	441,783	6446	

Tabela 7. Distância da Seqüência Principal (D)

Sistema	Média	Desvio Padrão	Máximo	Observação
C	**0,240**	0,240	0,960	Os números obtidos foram condizentes com as expectativas, a menor média da distância da seqüência principal foi a do projeto C, o único construído com TDD.
S	0,242	0,275	0,960	
E	0,285	0,242	0,750	
F	0,255	0,214	0,750	

4.5. Teste das Hipóteses

O teste das hipóteses pode ser realizado a partir da substituição direta dos resultados obtidos no estudo, compilados na Tabela 8.

Teste da Hipótese Nula (**H0**):

$((CC_C > CC_F) \wedge (CC_C > CC_S) \wedge (CC_C > CC_E)) = FALSO$

$((CBO_C > CBO_F) \wedge (CBO_C > CBO_S) \wedge (CBO_C > CBO_E)) = VERDADEIRO$

$((LCOM_C > LCOM_F) \wedge (LCOM_C > LCOM_S) \wedge (LCOM_C > LCOM_E)) = VERDADEIRO$

$((D_C > D_F) \wedge (D_C > D_S) \wedge (D_C > D_E)) = FALSO$

[2] Borland Together. Disponível em: http://www.borland.com/downloads/download_together.html

[3] JavaNCSS - A Source Measurement Suite for Java: <http://www.kclee.de/clemens/java/javancss/>

[4] Metrics v. 1.3.6: http://metrics.sourceforge.net/

[5] Dependency Finder: http://depfind.sourceforge.net/

Portanto, **H0**: FALSO∨VERDADEIRO∨VERDADEIRO∨FALSO = **FALSA**

Teste da Hipótese Alternativa 1 (**H1**):

$((CC_C \leq CC_F) \vee (CC_C \leq CC_S) \vee (CC_C \leq CC_E)) = VERDADEIRO$

$((LCOM_C \leq LCOM_F) \vee (LCOM_C \leq LCOM_S) \vee (LCOM_C \leq LCOM_E)) = FALSO$

$((CBO_C \leq CBO_F) \vee (CBO_C \leq CBO_S) \vee (CBO_C \leq CBO_E)) = FALSO$

$((D_C \leq D_F) \vee (D_C \leq D_S) \vee (D_C \leq D_E)) = VERDADEIRO$

Portanto, **H1**: VERDADEIRO∨FALSO∨FALSO∨VERDADEIRO = **VERDADEIRA**

Tabela 8. Teste das Hipóteses

	SISTEMAS				HIPÓTESES	
Metricas	**C**	**S**	**E**	**F**	**H0**	**H1**
CC	**1,756**	2,242	1,832	2,664	Falso	**Verdadeiro**
CBO	9,52	5,641	**0,992**	8,622	**Verdadeiro**	Falso
LCOM	153,248	**66,898**	119,249	123,279	**Verdadeiro**	Falso
D	**0,24**	0,242	0,285	0,255	Falso	**Verdadeiro**

Conclui-se que, no contexto da DTI, considerando amostra limitada de projetos, a **qualidade da estrutura interna do código fonte dos sistemas desenvolvidos com TDD não é menor que a daqueles que foram desenvolvidos sem TDD**.

Embora tenha sido possível refutar a hipótese nula, é prudente realizar estudos adicionais em larga escala para minimizar as diversas influências, por exemplo, as diferenças em termos de tamanho do sistema, complexidade, experiência e tamanho da equipe, ferramentas utilizadas e pressões de cronograma. A próxima seção apresenta uma breve discussão sobre alguns dos fatores que influenciaram o estudo.

4.6. Fatores que Influenciaram o Estudo

Heterogeneidade das Equipes: As equipes alocadas a cada um dos sistemas avaliados diferem consideravelmente em termos de tamanho e experiência. No tocante a experiência dos desenvolvedores, a equipe do projeto C era a que possuía menos experiência (experiência média em programação menor que dois anos). No projeto E o desenvolvedor possuía experiência de mais de cinco anos. Já o projeto S também contou com programador inexperiente (menos de dois anos experiência).

Pressões de cronograma: Dos projetos que foram comparados apenas o C e F possuiam forte compromisso com prazos. Ou seja, os demais sistemas foram desenvolvidos sem ter data fixada para entregar o produto. A pressão exercida pela gerência de desenvolvimento e pelos gestores de negócio fez com que a equipe do projeto C abrisse mão de uma das práticas fundamentais do TDD, o *refactoring* (isto aconteceu em várias iterações a partir da iteração 12). Acredita-se que isto tenha sido refletido nos números relativos ao acoplamento (CBO).

Natureza dos sistemas: Os projetos comparados são bem diferentes em termos da sua natureza e da complexidade do domínio, o que dificultou a avaliação através das métricas extraídas. Fatores importantes como a complexidade das regras de negócio não foram considerados.

Decisões Arquiteturais: Algumas determinações arquiteturais também influenciaram nos números. Durante o desenvolvimento do projeto C havia uma determinação para que os objetos da camada de negócio e da camada de integração fossem obtidos a partir de um único local (Fábrica). Estas fábricas que centralizam a obtenção de vários objetos

distintos apresentam baixa coesão. Além disso, todas essas classes apresentam a média da complexidade ciclomática baixa, em torno de 1 (um), pois são classes muito simples.

5. Conclusão

Pode-se perceber que os cursos práticos foram fundamentais para suprimir a resistência inicial. A DTI também pode contar com um especialista no assunto que fazia parte do quadro permanente, este aspecto foi decisivo para o sucesso da iniciativa.

Os desenvolvedores que antes dos projetos estavam bastante céticos com relação ao desenvolvimento orientado a testes, agora não cogitam desenvolver sem a utilização do TDD. A despeito de algumas métricas de qualidade interna terem sido desfavoráveis todos os membros da equipe afirmaram que se fossem desenvolver outro sistema o fariam tendo os testes como guia. Essa confiança confirma um dos benefícios citados na literatura: a segurança trazida pelos testes. No projeto Cadsinc foram freqüentes as mudanças nas regras de negócio, o que acarretou modificações constantes no código. Neste caso, os testes propiciaram a segurança necessária para que as modificações fossem realizadas.

É preciso ser cauteloso ao realizar qualquer análise sobre a aplicação de TDD na SEFAZ. Vários benefícios foram percebidos, mas não foram comprovados através de medições. Por isso, a gerência da DTI decidiu repetir a experiência em outros projetos. Espera-se coletar dados sobre a manutenibilidade destes sistemas e também sobre a produtividade no desenvolvimento e, a cada experiência, avaliar os efeitos da utilização de TDD.

Referências

Astels, D. (2007), “A New Look At Test-Driven Development”. Disponível em: <http://www.daveastels.com/files/BDD_Intro.pdf>. Acesso em 19 nov 2007.

Beck, K. (1999) “Extreme Programming Explained: Embrace Change”, Addison-Wesley

Beck, K. (2002) “Test Driven Development: By Example”, Addison-Wesley

Basili, V.R.; Caldiera, G.; Rombach, H. D. (1994), “The Goal Question Metric Approach”. In: Encyclopedia of Software Engineering 1. pp. 528-532.

Chidamber, S.; Kemerer, C. (1994), “A Metrics Suite for Object Oriented Design”. In: IEEE Transaction on Software Engineering. 20., pp. 476-493

Geras, A, Smith, M., Miller, J. (2004) “A Prototype Empirical Evaluation of Test Driven Development”, 10th International Symposium on Proceedings of the Software Metrics, IEEE Computer Society, pp. 405 - 416, Chicago.

George, B., Williams, L. (2003) “An Initial Investigation of Test Driven Development in Industry”, Proceedings of the 2003 ACM Symposium on Applied Computing, pp. 1135 – 1139, ACM Press, Melbourne, Florida.

Janzen, D., Saiedian, H. (2005) "Test-Driven Development: Concepts, Taxonomy, and Future Direction”. IEEE Computer, September, pp. 43-50.

Martin, R. C., (2003) “Agile Software Development, Principles, Patterns, and Practices”, Prentice-Hall

Mills, H. D., Dyer, M., Linger, R. C, (1987) "Cleanroom Software Engineering", IEEE Software, pp. 19-25

Müller, M., Hagner, O. (2002) “Experiment about Test-first programming”, IEE Proceedings Software, Vol. 149, Issue 5, pp. 131- 136

Watson, A. H., McCabe, T. J. (1996) "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric", National Institute of Standards and Technology Special Publication 500-235, 123p.

Williams, L., Maximilien, E. M., Vouk, M. (2003) “Test driven development as a defect-reduction practice”, 14th International Symposium on Software Reliability Engineering, pp 34 – 45. IEEE Computer Society, Denver, Colorado.