JAILTON JUNIOR DE SOUSA COELHO

MODELO DE DINÂMICA DE SISTEMAS PARA APOIO A DECISÕES NO PROCESSO DE INSPEÇÃO DE SOFTWARE

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS - BRASIL
2013

Ficha catalográfica preparada pela Seção de Catalogação e Classificação da Biblioteca Central da UFV

T
C672m
2013

Coelho, Jailton Junior Sousa, 1988-
Modelo de dinâmica de sistemas para apoio a decisões no processo de inspeção de software / Jailton Junior Sousa Coelho. – Viçosa, MG, 2013.
x, 106 f. : il. (algumas color.) ; 29 cm.

Inclui anexo.
Inclui apêndices.
Orientador: José Luis Braga.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f. 102-106.

1. Software - Inspeção. 2. Dinâmica. 3. Tolerância a falha (Computadores). I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. II. Título.

CDD 22. ed. 005.12

JAILTON JÚNIOR DE SOUSA COELHO

MODELO DE DINÂMICA DE SISTEMAS PARA APOIO A DECISÕES NO PROCESSO DE INSPEÇÃO DE SOFTWARE

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae.*

APROVADA: 07 de Novembro de 2013.

Dedico essa dissertação à Deus, aos meus pais
Chinelo e Netinha
e à minha namorada Natália Rezende

AGRADECIMENTOS

Agradeço profundamente à Deus por todas as coisas que Ele tem feito em minha vida, por mais essa conquista e por todas as coisas que Ele ainda irá fazer por mim. Sei que sem a ajuda dEle tudo seria muito mais difícil.

Agradeço aos meus pais Chinelo e Netinha por todo carinho, amor, compreensão e incentivo. Amo muito vocês.

Agradeço a todos os meus amigos do curso de computação de 2007, que tornaram meus anos de universidade bem divertidos. Dá-lhe CAVALO DE MAMINHA!

Agradeço a todos que passaram pela 411, tornando meus anos de IC bastante alegres.

Agradeço à minha namorada Natália por todo carinho. Você realmente mudou minha vida. Te amo muito.

Agradeço a todos os moradores da república.

Agradeço ao meu orientador Zé Luis por todo apoio dado e também pelos “dias de pizza”.

Agradeço a todos do DPI, que de alguma forma contribuiu para a realização desse trabalho.

À Universidade Federal de Viçosa.
À CAPES pelo apoio financeiro.

BIOGRAFIA

JAILTON JUNIOR DE SOUSA COELHO, filho de Jailton Santos Coelho e Luzinete Sousa Coelho, brasileiro, nascido em 7 de Novembro de 1988 na cidade de Eunápolis, no estado da Bahia.

No ano de 2007, após concluir o ensino médio na cidade de Eunápolis, ingressou no curso de graduação em Ciência da Computação na Universidade Federal de Viçosa, concluído no ano de 2011.

Em 2011, foi aprovado na seleção do programa de pós-graduação do Departamento de Informática – DPI, onde, em Março de 2011, iniciou o mestrado em Ciência da Computação na Universidade Federal de Viçosa – UFV, defendendo sua dissertação em Novembro de 2013.

SUMÁRIO

LISTA DE FIGURAS

Introdução

Artigo I

Artigo II

Artigo II

Artigo IV

Artigo V

LISTA DE TABELAS

Artigo IV

Artigo V

RESUMO

COELHO, Jailton Junior de Sousa, M. Sc., Universidade Federal de Viçosa, Fevereiro de 2013. **Modelo de dinâmica de sistemas para apoio a decisões no processo de inspeção de Software**. Orientador: José Luis Braga.

Reparar um defeito de software pode custar até 100 vezes mais caro, caso ele não seja encontrado o mais próximo possível de onde foi cometido. A inspeção de software é uma técnica que pode ser usada para ajudar a detectar defeitos nas primeiras fases do processo, evitando que esses defeitos sejam propagados para as fases seguintes. O custo/benefício de inspeções pode se tornar bastante significativo, se as inspeções forem realizadas de forma eficiente. Por ser influenciada por muitos fatores de qualidade, a análise do contexto da inspeção como um todo pode se tornar complexa. Gerentes de projeto deixam de utilizar a inspeção com dúvida dos reais benefícios que ela pode gerar. O objetivo desse trabalho é criar um modelo de dinâmica de sistemas, que é uma técnica descritiva utilizada para modelagem e simulação de sistemas, envolvendo variáveis que influenciam fortemente na eficiência da inspeção. Os níveis de influência das variáveis que fazem parte do modelo são quantificados com base em experimentos reais ou empíricos disponibilizados na literatura, tornando os resultados do modelo próximos do que seria obtido no mundo real. O modelo permite reproduzir cenários nos quais seria caro ou perigoso experimentar na realidade, sendo possível analisar os impactos que a inspeção pode trazer no processo de desenvolvimento.

ABSTRACT

COELHO, Jailton Junior de Sousa, M. Sc., Universidade Federal de Viçosa, Fevereiro de 2013. **System dynamics model for decision support on the software inspection process**. Adviser: José Luis Braga.

Repairing a defect in late phases of software development can be a hundred times more expensive than finding and fixing it during the requirements and design phase. Software inspection is a technique that may be used to aid in the identification of defects during early stages of the process and avoid propagation of such defects to the next phases. The cost-benefit of inspections may be significant if they are efficiently performed. Since this process is affected by several quality factors, the analysis of the overall context of inspection may become complex. Project managers are reluctant in introducing inspection due to uncertainty regarding its real benefits. This paper presents a system dynamics model, which is a descriptive technique for systems modeling and simulation and involves several variables that strongly influence inspection efficiency. The influence levels of model variables are quantified with basis on real or empirical experiments reported in literature, in order to approximate model results to values that would be obtained in the real world. The model allows the reproduction of scenarios without having the costs and facing the risk of a real project implementation. Therefore, it enables the analysis of the impacts of inspections on the software development process.

1 INTRODUÇÃO

A qualidade de um produto de software não pode ser garantida após a finalização do mesmo, sendo, portanto, um aspecto que deve ser acompanhado e garantido durante todo o processo de desenvolvimento (RUSSEL, 1991). A participação humana é um fator presente nos projetos de software, o que justifica a ocorrência de defeitos nos sistemas de software mesmo quando as melhores técnicas e procedimentos são empregados no seu desenvolvimento (KNIGHT, 1993).

Em torno de 40 a 50% do esforço no desenvolvimento de software é gasto em retrabalho e grande parte desse esforço é gasto com a remoção de defeitos, exigindo, em algumas situações, mudanças significativas na arquitetura do projeto (SHULL, 2002). O custo para remover um defeito de software aumenta à medida que o processo de desenvolvimento progride para os estágios finais (SHULL, 2002). Encontrar e corrigir um defeito de software pode ser até 100 vezes mais caro, caso ele não seja encontrado próximo da fase em que foi cometido no processo de desenvolvimento (SHULL, 2002). (LAITENBERGER, 2000) considera defeito como qualquer desvio nas propriedades de qualidade do software, o que inclui o não atendimento aos requisitos elicitados.

Atividades de prevenção e detecção de defeitos devem ser introduzidas ao longo de todo o processo de desenvolvimento com o objetivo de diminuir os custos de correção nas fases finais. Alocar recursos para a realização de revisões de artefatos de software, tão logo estejam finalizados, pode contribuir para reduzir o retrabalho e melhorar a qualidade dos produtos (FAGAN, 1986). Encontrar um defeito o mais cedo possível pode gerar uma enorme contribuição para o projeto, do ponto de vista econômico e de qualidade.

A inspeção de software é um tipo específico de revisão que pode ser aplicado em artefatos de software e possui um processo de detecção de defeitos rigoroso e bem definido (PORTER, 1995). Segundo (LAITENBERGER, 2000), inspeções têm a função de detectar defeitos antes que a fase de teste seja iniciada, contribuindo para a melhoria da qualidade geral do software em relação ao seu orçamento e tempo total

de desenvolvimento. Apesar desses benefícios, em geral, os gerentes de projetos não adotam a inspeção no processo de desenvolvimento de software devido ao receio de não obterem um retorno satisfatório (FAGAN, 1986), que decorre da falta de ferramentas para auxiliar na obtenção de uma estimativa do retorno que será obtido com a alocação de recursos para realizar tais atividades.

Esse trabalho apresenta um modelo de dinâmica de sistemas que permite avaliar os impactos da alocação de recursos para a realização de inspeções em várias fases do processo de desenvolvimento de software. O modelo permite simular e analisar os resultados da política gerencial quanto à realização de inspeções antes de aplicá-la em um projeto real. O modelo pode ser usado como uma ferramenta de apoio à decisão permitindo simular diversos cenários de acordo com as características da empresa, equipe e projeto, sem que seja necessário realizar experimentos em projetos reais, o que seria caro e demandaria bastante tempo.

No modelo foram incluídas variáveis que influenciam diretamente na qualidade do processo de inspeção, bem como no retorno obtido ao alocar recursos para a realização dessas atividades, dentre as quais se tem: tamanho da equipe de inspetores, técnica de leitura, nível CMM da empresa, tipo e tamanho do documento, entre outras. Essas variáveis e os relacionamentos de influência existentes entre as mesmas foram identificados e calibrados com base em experimentos reais ou empíricos publicados na literatura, o que garante a consistência do modelo e o torna coerente com os conhecimentos difundidos pela Engenharia de Software.

1.1 O Problema, sua Importância e Justificativa

A tomada de decisão é uma atividade importante e complexa na engenharia de software. Normalmente é um processo não sistemático, uma vez que tipicamente se baseia em experiência pessoal, sem o uso de modelos explícitos. O desenvolvimento de software é um processo onde cada pessoa envolvida constantemente toma decisões, tanto técnicas quanto gerenciais. Um gerente de projetos, por exemplo, tem um grande conjunto de fatores a considerar, várias atividades para planejar e controlar e diferentes decisões a tomar (BARRETO, 2005).

A capacidade de enxergar e compreender o sistema como um todo é fundamental nas decisões. Em projetos de software, os gestores, quando buscam visualizar de forma integrada todos os fatores de influência do seu processo de desenvolvimento, terão maior capacidade de tomar decisões promissoras. No geral,

quando os gestores não possuem esta visão do sistema como um todo, eles tomam decisões reativas, considerando apenas o problema presente, sem relacioná-lo com o seu ambiente, suas variáveis e demais fatores correlacionados (SENGE, 1990). Isso se torna um risco considerável no projeto, visto que decisões gerenciais tomadas de maneira equivocada podem comprometer todo um projeto levando a grandes prejuízos e, mais do que isso, podem levar à produção de softwares com falhas críticas, que podem significar grandes perdas no futuro, tanto financeiras como até mesmo de vidas humanas (PETERSON, 1996). Logo, é indispensável o uso de ferramentas que auxiliem os gerentes em suas tomadas de decisão, identificando os riscos antes deles se materializarem (HERMSDORF, 2011).

A importância de um bom gerenciamento de projeto de software é comprovada por meio de dados mostrados a seguir: 75% dos projetos de software falham, sendo sua principal causa o pobre gerenciamento. (CAVALCANTE, 2005) revela também que 31% dos projetos de software são cancelados antes de sua conclusão, e 53% excedem em mais de 50% suas previsões de prazo e custo. Os principais motivos para os problemas apresentados estão relacionados a falhas no gerenciamento de software durante a fase de desenvolvimento.

A inspeção é um tipo particular de revisão que contribui para garantir a qualidade do produto de software. Todas as etapas do processo de desenvolvimento de software são suscetíveis à incorporação de defeitos, que podem ser detectados pela inspeção e posteriormente removidos. É importante destacar que quanto mais cedo esses defeitos forem removidos, menor será o custo de desenvolvimento e manutenção do produto. A inspeção, quando realizada no início do desenvolvimento do software, leva à detecção de 60% a 90% dos defeitos potenciais em um projeto de software (BOEHM, 2001).

Embora muitas organizações de software realizem revisões, a forma como as revisões são realizadas ainda é pouco sistematizado, pouco conhecimento da área de inspeções de software é utilizado e assim o potencial raramente é explorado. Este argumento é baseado em três observações: (1) revisões raramente cobrem sistematicamente as fases de desenvolvimento de software (40% realizam inspeções em requisitos e 30% realizam inspeções em código), (2) muitas vezes a atividade de detecção de defeitos não é realizada sistematicamente (cerca de 60% não conduzem uma atividade de detecção de defeitos regularmente), e (3) cerca de 40% das

organizações não coletam dados e 18% coletam dados e não os utilizam em suas análises (KALINOWSKI, 2004).

Variáveis como o número e experiência dos inspetores, técnica de leitura, tipo e tamanho do documento, entre outras, influenciam diretamente na qualidade do processo de inspeção. Não gerenciar estas variáveis no processo de inspeção contribui para a ineficiência do processo, uma vez que estas variáveis influenciam fortemente na qualidade do processo de inspeção (DAIBERT, 2010).

Dada a importância do processo de inspeção e a complexidade das relações entre as variáveis dinâmicas envolvidas nessa fase do processo de desenvolvimento de software, torna-se indispensável o uso de ferramentas que permitam enxergar o contexto como um todo (AMBRÓSIO, 2011). A dinâmica de sistemas (MADACHY, 2007; STERMAN, 2000) é uma dessas ferramentas, e será explorada neste trabalho para permitir uma melhor visualização do contexto e auxiliar na tomada de decisões relativas ao processo de inspeção de softwares (HERMSDORF, 2011). A utilização da dinâmica de sistemas, no contexto da engenharia de software, busca contribuir para a simulação de ambientes de desenvolvimento. Ela também serve como um modelo de apoio à decisão para os gestores, possibilitando melhorias na qualidade do projeto (DAIBERT, 2010).

1.2 Hipótese

A utilização de uma ferramenta que aborde o processo de inspeção de software, com base em modelos de simulação de dinâmica de sistemas, possibilita aos gerentes tomarem decisões mais eficientes relativas aos projetos de desenvolvimento de software.

1.3 Objetivos

O objetivo geral deste trabalho é construir um modelo de dinâmica de sistemas, que permite avaliar os impactos da utilização de inspeção em várias etapas do ciclo de desenvolvimento de projetos de software.

Os objetivos específicos relacionados são:

- Selecionar as principais variáveis dinâmicas envolvidas na qualidade do processo de inspeção;
- Elaborar diagramas de influência com o objetivo simplificar a visualização das relações entre as variáveis envolvidas;

- Mapear os diagramas de influência para um modelo estoque-fluxo da dinâmica de sistemas;
- Validar o modelo com dados da literatura que comprovem a sua consistência, e estabeleçam a viabilidade do seu uso no apoio à tomada de decisão;
- Construir uma ferramenta de apoio que permita o uso efetivo dos modelos de dinâmica de sistemas por empresas de software.

1.4 Método de Desenvolvimento do Trabalho

Desde os estágios iniciais da pesquisa, buscou-se realizar um trabalho embasado em informações comprovadas e publicadas na literatura. Isso garante uma maior confiabilidade aos resultados alcançados e às conclusões obtidas, e também torna explícito o contexto em que o trabalho está inserido e o problema que o mesmo pretende resolver.

Visando alcançar os resultados que satisfaçam os objetivos do trabalho, foram realizadas as atividades a seguir de maneira iterativa, que é um método proposto por (AMBRÓSIO, 2011).

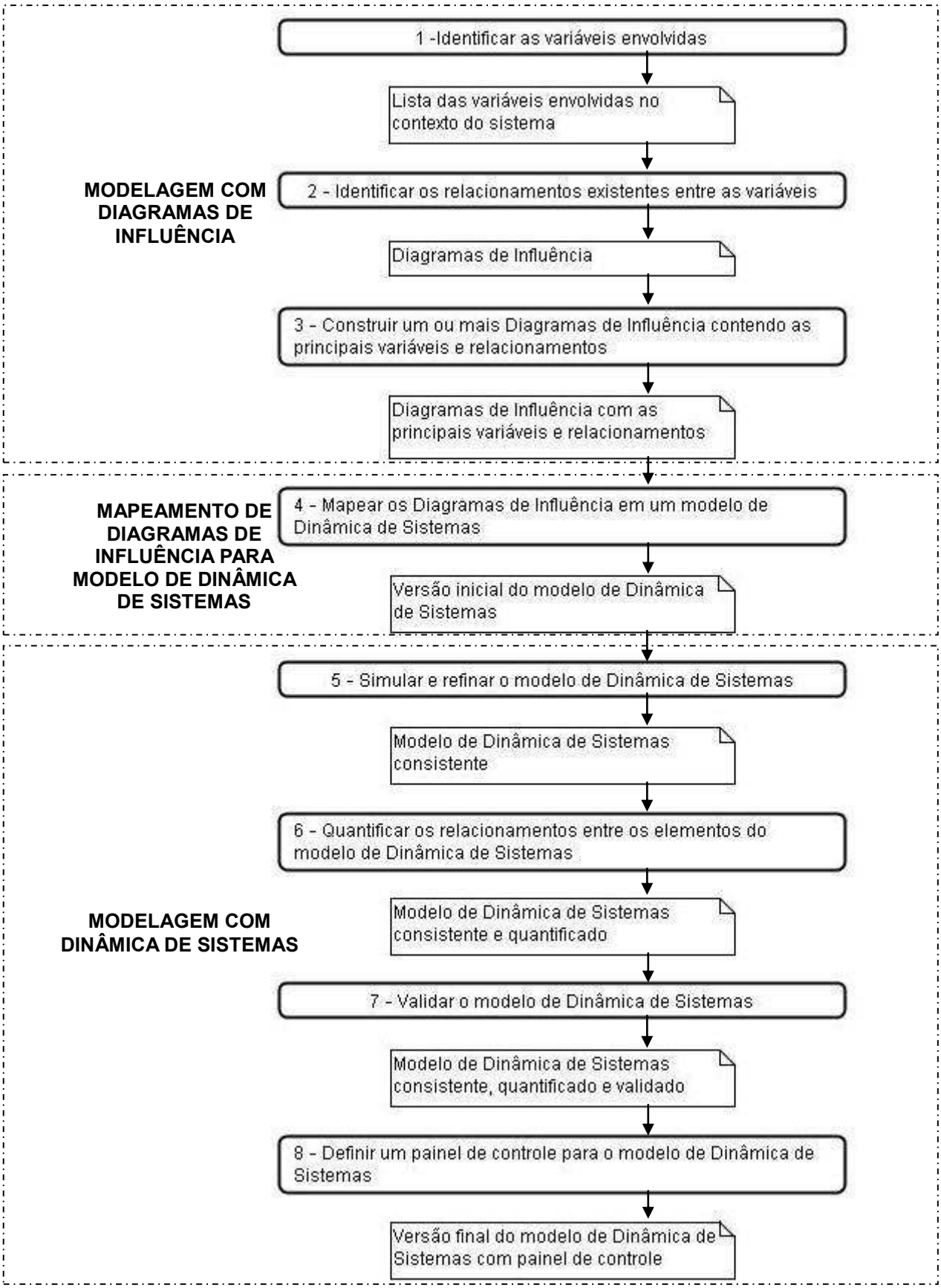

Figura 1. Método para a construção de modelos de dinâmica de sistemas.

1.5 Evolução dos modelos

No total foram produzidos 4 modelos, sendo que foram gerados artigos de apenas 3 deles.

O primeiro modelo produzido (Fig. 2) tem a finalidade de identificar os requisitos que são elicitados com erros pela técnica de entrevista. Este modelo possui diversas limitações, dentre elas, a dependência do modelo produzido por Hermsdorf (HERMSDORF, 2011).

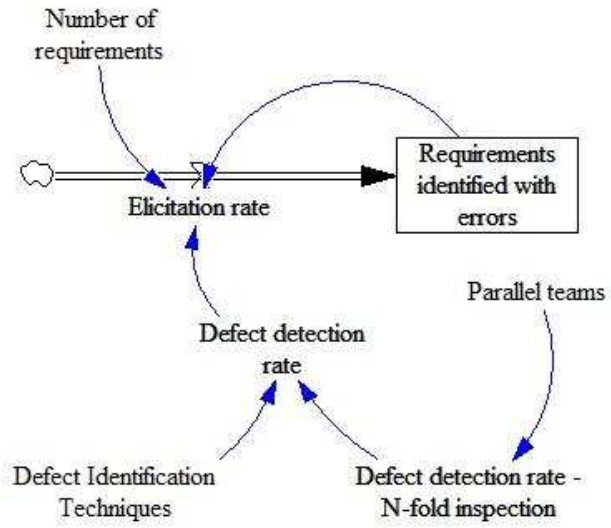

Figura 2. Primeiro modelo produzido.

No modelo apresentado na Fig. 2 é possível a simulação de 6 diferentes técnicas de inspeção: *Formal Specification Method, N-fold, Ad Hoc, Checklist, Scenario* e *Perspective*. E para a técnica *N-fold* é permitida a simulação de até 15 equipes de inspetores paralelas.

O segundo modelo produzido possui o mesmo objetivo que o primeiro, a identificação dos requisitos que são elicitados com erros. Este modelo também possui como limitações a dependência do modelo produzido por Hermsdorf (HERMSDORF, 2011). Entretanto ele difere do primeiro modelo já que há a possibilidade de simulação de 4 tipos de técnicas de elicitação.

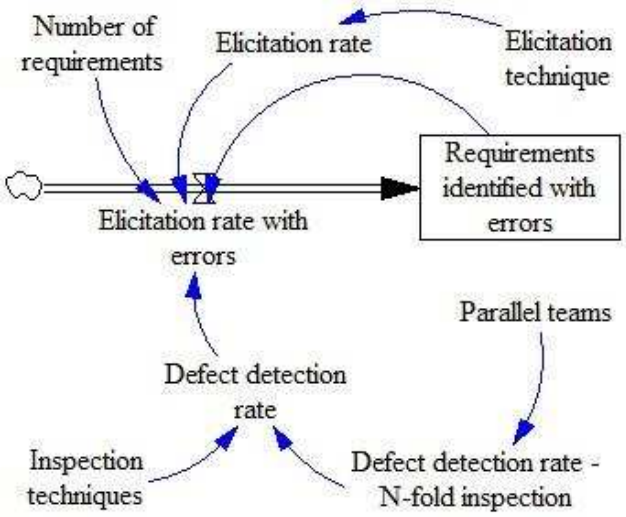

Figura 3. Segundo modelo produzido.

Por meio do modelo apresentado na Fig. 3 é permitida a simulação das seguintes técnicas de elicitação: Interviewing, Laddering, Sorting e Protocol Analysis. Todas essas técnicas possuem diferentes taxas de elicitação, gerando diversos tipos de requisitos com erros. Portanto, através do modelo é possível simular o número de requisitos com erros que seriam identificados caso fosse aplicada alguma técnica de inspeção de software.

O terceiro modelo produzido possui o objetivo de identificar os tipos de requisitos com erros que seriam encontrados caso fossem aplicadas técnicas de inspeção de software ao processo de elicitação de requisitos. O modelo parte do pressuposto de que cada técnica de inspeção é especializada na identificação de diferentes tipos de erros.

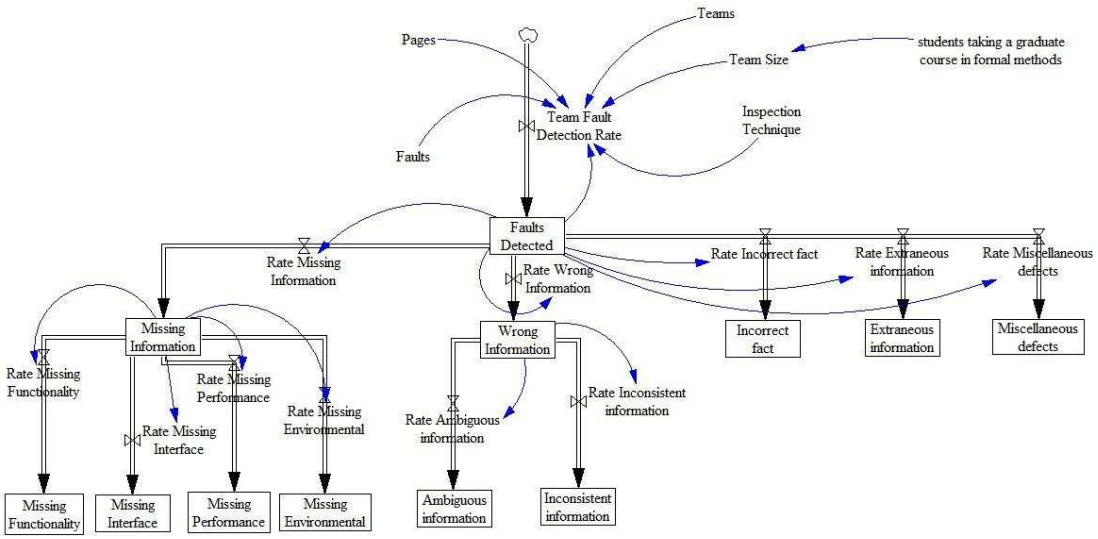

Figura 4. Terceiro modelo produzido.

Através do modelo apresentado na Fig. 4 é permitida a simulação das técnicas de leitura: *Formal Specification Method*, *N-fold*, *Ad Hoc*, *Checklist*, *Scenario* e *Perspective*. Este modelo permite também simulações para documentos de requisitos com diferentes números de páginas e erros por página, e é possível definir a experiência dos inspetores. Entretanto, o modelo acabou perdendo força pelo fato das fontes literárias divergirem quanto às probabilidades dos tipos de requisitos com erros que são identificados.

O quarto e último modelo produzido, apresentado na Fig. 5, tem o objetivo de simular o processo de inspeção utilizando como parâmetros de entrada os fatores de grande influência no processo segundo (LAITENBERGER, 1991). Através do modelo é possível estimar o total de defeitos que serão detectados e não detectados, e também o custo necessário para realização da atividade.

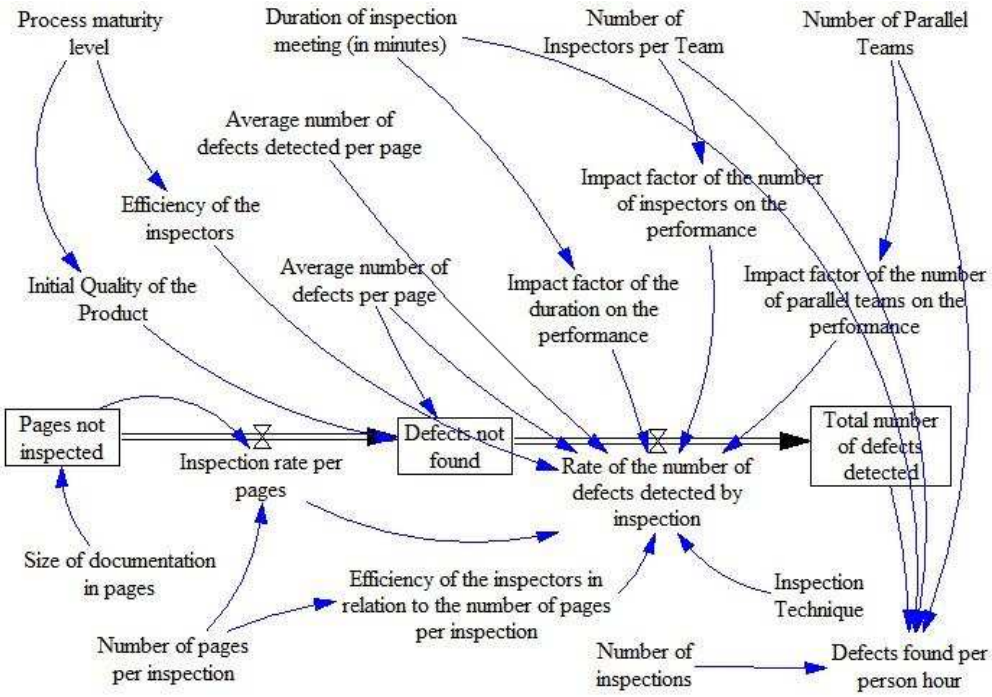

Figura 5. Quarto modelo produzido.

Por meio do modelo apresentado na Fig 5 há a possibilidade de simulação de 5 técnicas de leitura, que são as mais utilizadas pelas organizações: Ad Hoc, LBCh, LBPe, LBCe e N-fold. Através do modelo é permitida também a escolha do tamanho da equipe de inspetores, técnica de leitura, nível CMMI da empresa, tipo e tamanho do documento, entre outras.

1.6 Organização desse documento

Esta dissertação está organizada no formato de coletânea de artigos, ao invés do formato tradicional de dissertação ou tese.

O Capítulo 1 apresenta o problema, sua importância, os objetivos da pesquisa e o método utilizado para construção do modelo.

O Capítulo 2 é composto de cinco artigos resultantes da pesquisa realizada. O Artigo I (Seção 2.1) foi submetido ao CLEI 2012. O Artigo II (Seção 2.2) foi submetido ao JCC 2012. O Artigo III (Seção 2.3) foi submetido à revista Engenharia de Software Magazine. O Artigo IV (Seção 2.4) foi submetido ao ASSE 2013. Por fim, o Artigo V (Seção 2.5) foi submetido ao ACM SIGSOFT Software Engineering Notes.

A seguir estão relacionados as referências completas dos artigos que foram publicados:

- COELHO, J. J. S.; BRAGA, J. L. Apoio a decisões no processo de inspeção de software. Engenharia de Software Magazine, ISSN 1983-1277, Volume 5, edicao 57, p. 24-32, 29 mar. 2013.
- COELHO, J. J. S.; BRAGA, J. L; AMBRÓSIO, B. Otimizando recursos no processo de inspeção de software: uma abordagem utilizando simulação com dinâmica de sistemas. 14th Argentine Symposium on Software Engineering, 42JAIIO, Córdoba, Argentina, Setembro 16-17, 2013.
- COELHO, J. J. S.; BRAGA, J. L; AMBRÓSIO, B. G. System dynamics model for simulation of the software inspection process. ACM SIGSOFT Software Engineering Notes, ISSN 0163-5948, Volume 38, Issue 5, p. 1-8, Setembro de 2013.

Finalmente, o Capítulo 3 apresenta as conclusões obtidas e as principais contribuições disponibilizadas por esse trabalho, destacando as decisões gerenciais suportadas pelo modelo construído. São disponibilizadas também sugestões para trabalhos futuros, ressaltando a possibilidade de novas variáveis e relacionamentos serem adicionados no modelo.

Ao final do documento é apresentado um apêndice contendo um glossário com a definição de todos os elementos do modelo de dinâmica de sistemas construído, que foram descritos nos artigos.

2 ARTIGOS

2.1 Apoiando decisões em análise de requisitos de software: uma abordagem utilizando simulação com dinâmica de sistemas

Jailton Jr. De S. Coelho , José Luis Braga , Bernardo Giori Ambrósio

ABSTRACT

Decision making during requirements analysis is a very important step in the requirements engineering process. Problems with requirements should be detected and resolved soon, so minimizing errors injected in the development phase. An important decision regards techniques for error detection to be used in the process. This paper presents and discusses six of those techniques, analyzed through the use of simulation based on scenarios and using system dynamics. Main variables present dynamic behavior, leading decisions towards system dynamics. The results presented are partial and will be refined in subsequent steps.

Keywords: Gerência de Projetos de Software; Dinâmica de Sistemas; Elicitação de Requisitos.

1 INTRODUÇÃO

Obter qualidade nos projetos de desenvolvimento de software não é uma tarefa trivial. Muitos fatores influenciam os objetivos de qualidade. Experiências apontam que a maioria dos problemas ocorre devido à falta de atenção durante a fase de requisitos e no acompanhamento da evolução dos requisitos de software ao longo do seu desenvolvimento (LEITE, 1997).

Grande parte dos problemas ocorridos nos projetos de software está relacionado aos requisitos. É muito comum um requisito especificado não refletir a real necessidade do cliente. Requisitos incompletos e/ou inconsistentes, mudanças em requisitos já especificados e a dificuldade de se obter um entendimento sem ambiguidades entre usuários e desenvolvedores são algumas das principais

dificuldades encontradas no processo. Esses problemas frequentemente geram retrabalho, atrasos no cronograma e aumento dos custos, trazendo insatisfação tanto para os desenvolvedores quanto para os clientes e usuários do software (BLASCHEK, 2012).

Muitos desses problemas nos requisitos poderiam ser evitados se as organizações fizessem uso de um processo de gestão de requisitos bem definido, controlado, medido e aprimorado. Entretanto, vê-se que na prática isso não é tão comum. Muitos profissionais não dão a importância devida a esta questão, o que acaba dificultando a ação dos gerentes de projetos.

É importante ressaltar o conceito de requisito de software, que é uma especificação de uma característica ou propriedade que o sistema deve ter ou fazer. Eles podem ser classificados como requisitos de negócios, funcionais, não funcionais, etc.

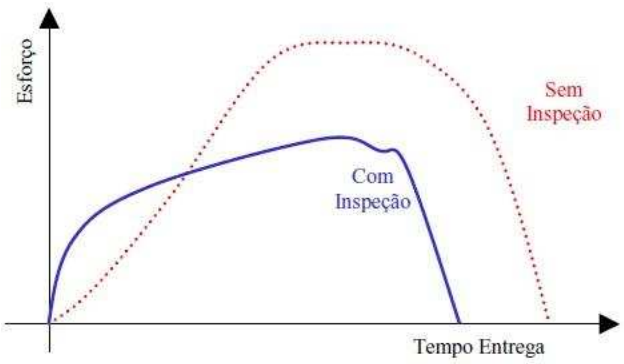

Figura 1. Importância da inspeção de erros em requisitos em projetos de software (PAGLIUSOA *et al*., 2003).

A partir da análise da Fig. 1, pode-se concluir que a inspeção dos artefatos produzidos durante a fase de requisitos é uma técnica efetiva e eficiente, podendo detectar diversos tipos de problemas em requisitos antes destes se tornarem base para o projeto e implementação do sistema, evitando o retrabalho (PAGLIUSOA *et al*., 2003).

A dinâmica de sistemas é uma técnica adequada para modelar e simular esse tipo de processo, pois ela permite levar em consideração as variações dinâmicas envolvidas nos processos modelados. A utilização dessa técnica permite a análise e compreensão de problemas e situações de maneira integrada e interconectada, deixando claras as relações existentes entre as variáveis de decisão (BASTOS, 2003).

Este trabalho descreve um projeto de pesquisa em andamento, que tem como objetivo simular o uso de diferentes técnicas de detecção de erros em requisitos, que

estão em uso corrente na engenharia de software. Resultados parciais são apresentados por meio de um modelo de dinâmica de sistemas que permite aos gerentes de projetos simular diversos cenários. Antes de executar uma simulação, os gerentes podem configurar qual técnica de identificação de erros em requisitos será considerada. Os gráficos gerados após a simulação permitem verificar qual técnica é mais adequada para um determinado tipo de projeto.

Na próxima seção serão apresentados o contexto e os trabalhos correlatos que foram tomados como base para o desenvolvimento deste trabalho. Na Seção 3 será apresentada a modelagem da atividade de análise de requisitos utilizando diagramas de influência (Seção 3.a), o modelo estoque-fluxo da dinâmica de sistemas (Seção 3.b) e o painel de controle utilizado para configuração dos cenários (Seção 3.c). As simulações realizadas e suas análises são discutidas na Seção 4. Já na Seção 5 é feita uma comparação entre as técnicas de detecção de erros em requisitos utilizadas nas simulações. Por último, as conclusões e os trabalhos futuros são apresentados na Seção 6.

2 CONTEXTO E TRABALHOS CORRELATOS

Este trabalho aborda a utilização da dinâmica de sistemas para modelar a fase de requisitos no processo de desenvolvimento de software, e mais especificamente a atividade de análise de requisitos, que é uma das primeiras atividades realizadas no processo de desenvolvimento de software. Entre os diversos trabalhos relacionados à aplicação de dinâmica de sistemas na modelagem de processos de desenvolvimento de software, destacam-se os trabalhos (ABDEL-HAMID, 1991) e (MADACHY, 2007). Em relação à modelagem focada na fase de requisitos, destacam-se os trabalhos (AMBROSIO, 2008) e (HERMSDORF, 2011).

Ambrósio (2008) propôs um modelo de dinâmica de sistemas que abrange as principais variáveis envolvidas na fase de extração de requisitos de projetos de software e descreve como essas variáveis podem se relacionar umas com as outras. Com esse modelo proposto, gerentes de projeto conseguem buscar diferentes alternativas para verificar e antever os impactos da materialização de riscos, tais como o turnover de pessoal, volatilidade dos requisitos, retrabalho devido a erros, falhas em planejamento, entre outros.

Já no modelo proposto por Hermsdorf (2011) é abordada a contextualização dos requisitos na atividade de elicitação, onde são analisadas as variáveis dinâmicas que impactam nessa atividade e que devem ser levadas em consideração para a tomada de decisões nessa etapa do processo de desenvolvimento. A partir desse modelo é possível verificar, por exemplo, como a experiência dos stakeholders envolvidos no projeto afeta a produtividade da elicitação, qual é a influencia do número de pessoas (elicitadores e 2 stakeholders) sobre o custo e a produtividade da equipe no projeto, etc.

Um complemento adicional e de grande importância que pode ser adicionado ao modelo proposto por Hermsdorf (2011) é a inclusão de técnicas de detecção de erros em requisitos. No presente trabalho serão abordadas as principais técnicas de identificação de erros em requisitos que impactam na atividade de análise de requisitos e que devem ser levadas em consideração para a tomada de decisões nessa etapa do processo de desenvolvimento.

Neste trabalho, os parâmetros estudados são relacionados à técnica de entrevista (BRAGA, 2004), que é a técnica mais utilizada na prática pelos analistas para realizar a elicitação de requisitos (DAVIS, 2006). Para identificação de erros em requisitos, foram investigadas as seguintes técnicas que são algumas das mais reportadas na literatura (LAMSWEERDE, 2000): Formal Specification Method, N-fold Inspection, Ad Hoc Reading, Checklist Based Reading Scenario Based Reading (Defect Based Reading), Perspective Based Reading.

Foram tomados como base os trabalhos (AHMED, 2009), (HERMSDORF, 2011) e (AURUM, 2002), que fazem uma revisão sistemática e uma agregação de resultados de diversos estudos sobre a técnica de elicitação e as técnicas de detecção de erros investigadas nesse trabalho.

3 MODELAGEM DA ATIVIDADE DE ANÁLISE DE REQUISITOS

O modelo de dinâmica de sistemas apresentado neste trabalho foi construído seguindo um método proposto por Ambrósio (AMBROSIO, 2008), constituído por três etapas subdivididas em um ou mais passos. A primeira etapa consiste na modelagem dos diagramas de influência; na segunda etapa, mapeiam-se os diagramas de influência em modelos de dinâmica de sistemas; e na última etapa, os

modelos de dinâmica de sistemas são refinados até que seja obtido um modelo finalizado.

Na primeira etapa, são identificadas as variáveis envolvidas no processo ou problema que se pretende modelar. Em seguida, identificam-se os relacionamentos existentes entre essas variáveis e por último, os diagramas de influência, que mostram as relações entre as variáveis, são construídos. Na segunda etapa os diagramas de influência são mapeados em modelos de dinâmica de sistemas. Na terceira e última etapa, os modelos são refinados e os relacionamentos entre seus elementos são quantificados, até que seja obtido um modelo finalizado e validado. Para facilitar a configuração de diferentes cenários, é recomendada a construção de um painel de controle que permite aos usuários definir os valores de alguns parâmetros do modelo antes de realizar as simulações.

O modelo de dinâmica de sistemas discutido neste trabalho supõe que a técnica de entrevista (BRAGA, 2004) é a adotada para a elicitação. Mantendo essa decisão, as simulações com o modelo permitem analisar as seguintes técnicas de detecção de erros em requisitos: Formal Specification Method, N-fold Inspection, Ad Hoc Reading, Checklist Based Reading, Scenario Based Reading (Defect Based Reading) e Perspective Based Reading. Os relacionamentos de influência das variáveis do modelo foram definidos de acordo com dados extraídos da literatura, tomando como base os estudos de (HERMSDORF, 2011) para definir a taxa de elicitação de requisitos ao utilizar a técnica de entrevista, e as pesquisas de (AHMED, 2009) e (AURUM, 2002) para quantificar os erros que são encontrados nos requisitos utilizando cada uma das técnicas investigadas.

3.1 Diagrama de Influência

Para entender melhor como as principais variáveis envolvidas no contexto do problema abordado neste trabalho se relacionam, foram utilizados os diagramas de influência. Um diagrama de influência com as principais variáveis envolvidas é apresentado na Fig. 2. Os arcos positivos (+) indicam que as variáveis de causa e efeito (início e ponta do arco, respectivamente) variam no mesmo sentido, ou seja, com tudo mais permanecendo constante, se a causa aumenta (diminui), o efeito também aumenta (diminui). O arco negativo (-) indica que causa e efeito alteram em

sentidos opostos, se a causa aumenta (diminui), o efeito sofre diminuição (aumenta) (FORRESTER, 1980).

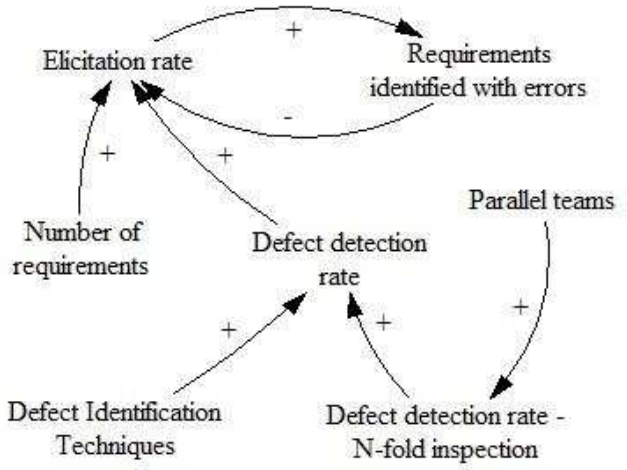

Figura 2. Diagrama de influência para a detecção de erros em requisitos na atividade de análise.

O número de requisitos afeta positivamente a taxa de elicitação, pois quando se necessita elicitar uma quantidade maior de requisitos, o esforço para encontrar novos requisitos no início dessa atividade é baixo, aumentando a taxa de elicitação. A técnica de identificação de erros em requisitos influencia positivamente a taxa de identificação de erros, pois quanto melhor for a técnica, maior será a taxa de identificação desses erros.

A taxa de detecção de erros afeta positivamente a elicitação dos requisitos porque quanto mais erros de requisitos forem identificados, menor será o retrabalho para consertar os requisitos com problemas, aumentando assim a taxa de elicitação.

A variável "Parallel teams" influencia positivamente a variável "Defect detection rate – N-fold inspection", pois quanto maior for o número de times alocados para a inspeção de erros nos requisitos através dessa técnica, maior será o número de erros detectados (LAMSWEERDE, 2000).

3.2 Modelo de Dinâmica de Sistemas

A partir das relações estabelecidas no diagrama de influência apresentado na Fig. 2, foi construído um modelo estoque-fluxo da dinâmica de sistemas. Para construir esse modelo foi utilizada a ferramenta Vensim (VENSIM, 2010), em sua versão gratuita para uso acadêmico.

O modelo é apresentado na Fig. 3 e possui os três principais tipos de elementos de um modelo de dinâmica de sistemas: estoques, fluxos e conversores. Os estoques (representado no modelo pelo elemento Requirements identified with errors) representam os recursos acumuláveis do sistema. Os fluxos (representado pelo elemento Elicitation rate) são funções que representam as decisões ou políticas da empresa com relação aos usos e acúmulos dos estoques ou recursos. E os conversores (demais elementos) são os elementos do modelo que exercem influência sobre os valores dos fluxos responsáveis pela variação dos estoques (AMBROSIO, 2008). O modelo apresentado na Fig. 3 foi obtido a partir do mapeamento das variáveis do diagrama de influência apresentado na Fig. 2 nos elementos estoques, fluxos e conversores da dinâmica de sistemas.

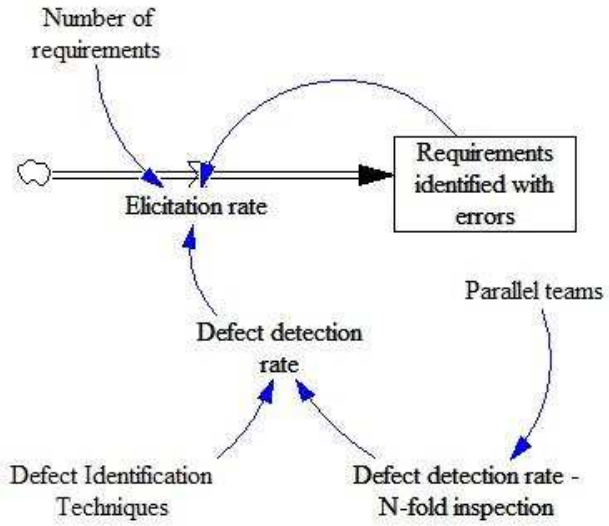

Figura 3. Modelo de dinâmica de sistemas para a detecção de erros em requisitos na atividade de análise.

3.3 Painel de Controle

O Painel de Controle apresentado na Fig. 4 foi construído com o objetivo de facilitar a interação dos usuários com o modelo proposto. Esse painel permite aos usuários definir os valores das seguintes variáveis para configurar um cenário para ser simulado: a) “Parallel teams”, que indica o número de times paralelos alocados para a técnica “N-fold Inspection”, e cujo valor pode variar de 1 a 15; b) “Number of requirements”, que indica o número de requisitos que serão elicitados, e cujo valor pode variar de 0 a 10000; c) “Defect Identification Techniques”, que indica qual técnica de detecção de erros em requisitos será considerada durante a simulação.

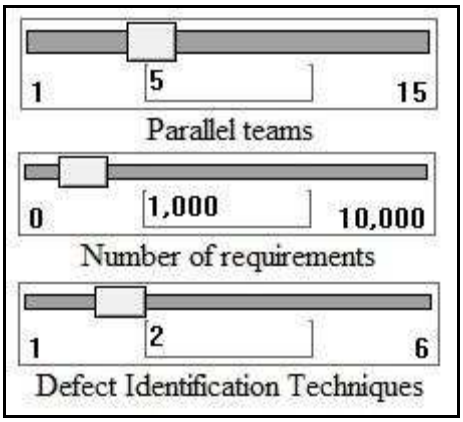

Figura 4. Painel de controle para configuração de cenários.

4 SIMULAÇÃO E ANÁLISE DOS RESULTADOS

Para executar as simulações com o modelo de dinâmica de sistemas apresentado na seção anterior, foram criados oito cenários que abrangem as seis técnicas de detecção de erros em requisitos discutidas nesse trabalho. Em todos os cenários simulados foi considerado que um elicitador engenheiro/analista e um stakeholder experiente trabalham na elicitação de 1000 requisitos durante um período de 30 horas de trabalho. Esta configuração foi escolhida baseando-se nos estudos realizados por Hermsdorf (2011) que prevê, para esta configuração, a ocorrência de 339 requisitos com erros entre os 1000 requisitos elicitados.

Os cenários de C1 a C8 utilizam a configuração mencionada anteriormente, mas diferem quanto à técnica de detecção de erros em requisitos utilizada. Para o cenário C1 foi utilizada a técnica Formal Specification Method; para os cenários de C2 a C4 foi usada a técnica N-fold Inspection; e os cenários C5, C6, C7 e C8 foram simulados utilizando as técnicas Ad Hoc Reading, Checklist Based Reading, Scenario Based Reading (Defect Based Reading) e Perspective Based Reading, respectivamente.

4.1 Formal Specification Method

A utilização de métodos formais é um meio satisfatório de garantir a qualidade dos requisitos, uma vez que esses métodos produzem especificações precisas, que podem ser rigorosamente validadas. Esses métodos de especificação podem ser utilizados em ambientes industriais, aplicações de controle de satélites, controle de tráfego, sistemas de automação de manufatura, sistemas de defesa, dentre outros.

Porém, eles ainda enfrentam obstáculos, principalmente no ambiente das pequenas e médias empresas. Isso se justifica pelo fato de eles necessitarem de um alto grau de entendimento, uma vez que muitos desses métodos baseiam-se em linguagens de especificação que empregam formalismos com forte base matemática e lógica (CARPENA, 1998).

O Modelo Formal de Requisitos possui um nível de abstração baixo e constitui a base formal para a utilização do método. Ele considera apenas o comportamento ideal do sistema (omitindo detalhes como precisão e tempo) e os requisitos podem ser expressos como conjuntos de funções sobre os estados das variáveis (CARPENA, 1998).

Métodos formais podem ser aplicáveis a qualquer projeto, e a melhora na especificação será refletida em um projeto executado de forma mais bem elaborada. Sistemas de missão crítica se beneficiam diretamente pela parte de verificação do método, porém todos os projetos se beneficiam de um documento de requisitos mais robusto (HALL, 1990).

A matemática usada não é difícil, e até pode-se dizer que é mais fácil do que a própria linguagem de programação. Usar qualquer ferramenta de computação requer algum nível de aprendizado; a notação formal não deve fugir a essa regra. Usar treinamento e consultoria em matemática discreta e na notação formal pode diminuir significativamente a curva de aprendizado (HALL, 1990).

Na realidade, adotar um método formal ajuda a especificar melhor o produto, e por isso, tende a encurtar o tempo total de implementação. Logo, mesmo passando-se mais tempo detalhando a especificação, o tempo programando será menor (HALL, 1990).

O resultado da simulação do cenário C1 com a técnica descrita é exibido na Fig. 5.

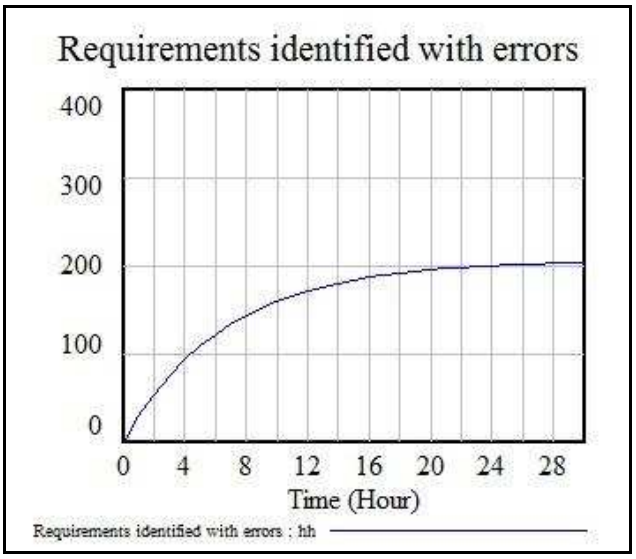

Figura 5. Resultado da simulação com a técnica Formal Specification Method.

Analisando-se o gráfico da Fig. 5 é possível perceber que uma grande parte de requisitos defeituosos foram identificados. Mais precisamente, foram detectados 60% dos 339 requisitos esperados com problemas.

4.2 N-fold Inspection

O método de inspeção formal é uma abordagem muito eficiente e eficaz segundo a literatura, porém pode exigir um alto investimento. Particularmente, no processo de análise de requisitos essa técnica pode ajudar a encontrar vários tipos de erros em requisitos, dentre eles requisitos inconsistentes ou incorretos. A execução dessa técnica consiste em replicar o processo de realização de inspeção formal para a avaliação da especificação de requisitos usando N equipes trabalhando em paralelo com um único moderador que é o responsável por coordenar e reunir os resultados.

A lógica por trás da técnica N-fold Inspection é que uma falha que não é supostamente encontrada por uma única equipe pode ser encontrada se várias equipes estão trabalhando em um mesmo artefato. Espera-se também que as equipes de inspeção encontrem defeitos ou falhas diferentes, contribuindo para que uma maior quantidade de defeitos nos requisitos seja encontrada (LAMSWEERDE, 2000).

Apesar desse método utilizar N equipes diferentes, nem sempre o resultado do trabalho será equivalente à soma do esforço de cada uma das equipes. Experiências apontam também que nem sempre há um grande volume de erros detectados quando se utiliza mais de uma equipe, pois muitas equipes podem acabar encontrando o mesmo erro durante a análise.

O melhor valor de N baseia-se em três fatores: a disponibilidade de equipes, o custo de cada equipe adicional e o custo potencial de não encontrar um defeito durante a inspeção. Estudos revelaram que a inspeção N-fold não depende do tipo e do tamanho do sistema a ser desenvolvido. Tais estudos sugerem ainda que equipes de inspeção e fiscalização com mais especialistas em cada equipe podem garantir um melhor desempenho (LAMSWEERDE, 2000).

Algumas das principais vantagens dessa técnica são: a sobreposição de detecção de falhas por equipes diferentes é mínima, a detecção de falhas é mais precoce e eficaz, e duas ou mais equipes de inspeção trabalhando no mesmo documento podem identificar mais falhas que uma única equipe. Já entre as desvantagens pode-se mencionar que algumas falhas podem não ser encontradas durante a inspeção, mesmo utilizando-se várias equipes, e que a inspeção de erros em requisitos realizada por múltiplas equipes pode tornar-se muito custosa se for realizada de forma ineficiente e mal planejada.

Os resultados das simulações para os cenários C2 com N = 1, C3 com N = 5 e C4 com N = 10, utilizando a técnica discutida nessa seção são exibidos nas Figs. 6, 7 e 8, respectivamente.

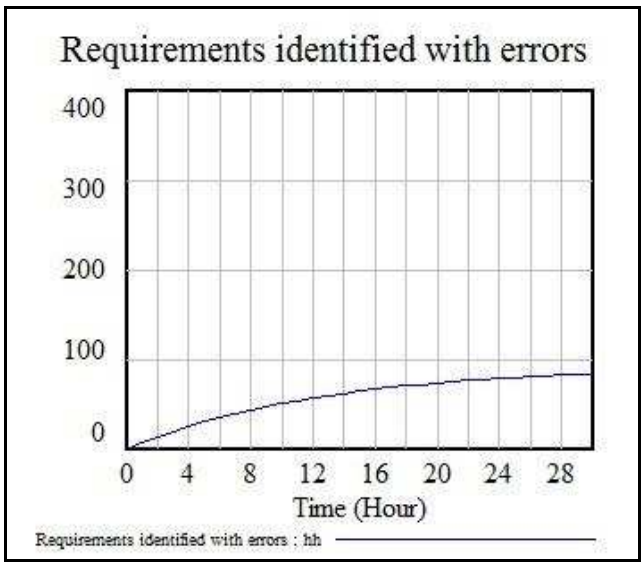

Figura 6. Resultado da simulação com a técnica N-fold Inspection para n = 1.

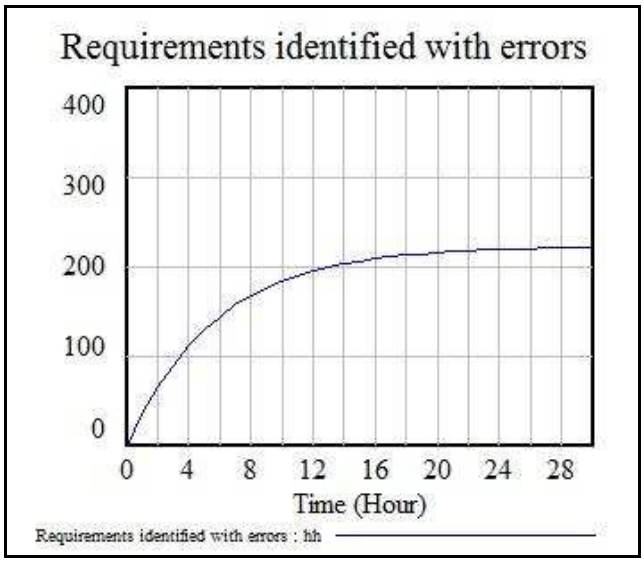

Figura 7. Resultado da simulação com a técnica N-fold Inspection para n = 5.

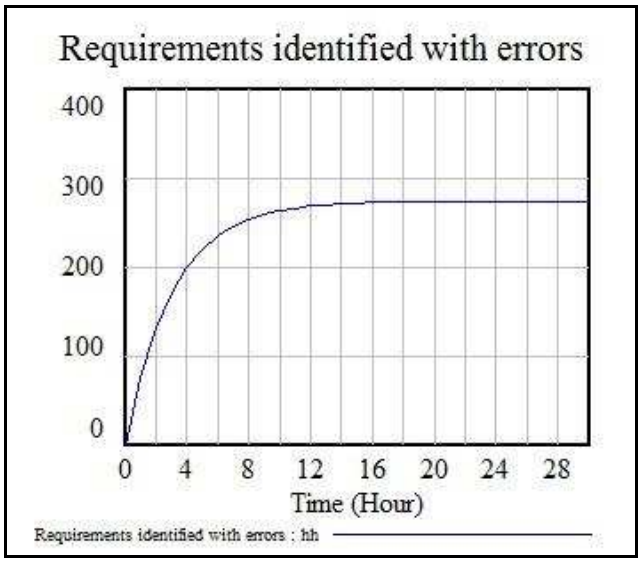

Figura 8. Resultado da simulação com a técnica N-fold Inspection para n = 10.

Analisando-se os resultados descritos pelos gráficos para cada valor de N, pode-se notar uma melhora significativa no número de requisitos com erros encontrados. Precisamente, 27%, 65% e 80% dos requisitos com erros esperados foram detectados, para os valores de N = 1, N = 5 e N = 10, respectivamente.

4.3 Ad Hoc Reading

A técnica de leitura ad-hoc é baseada principalmente no conhecimento e na experiência do inspetor, já que ela não fornece suporte técnico para apontar quais informações checar e como extrair defeitos nessas informações. Isto não supõe que as inspeções com esta abordagem não sigam critérios adequados, mas pode indicar

que os critérios e os métodos utilizados na análise ou leitura dos artefatos dependem do inspetor que as efetua (SAYAO, 2012).

Várias qualidades podem ser alcançadas em um documento de requisitos verificado por esta técnica, dentre elas estão: clareza, completude, consistência, corretude, funcionalidade, testabilidade e detalhamento (SAYAO, 2012).

O resultado da simulação para o cenário C5 com a utilização da técnica descrita é mostrado na Fig. 9.

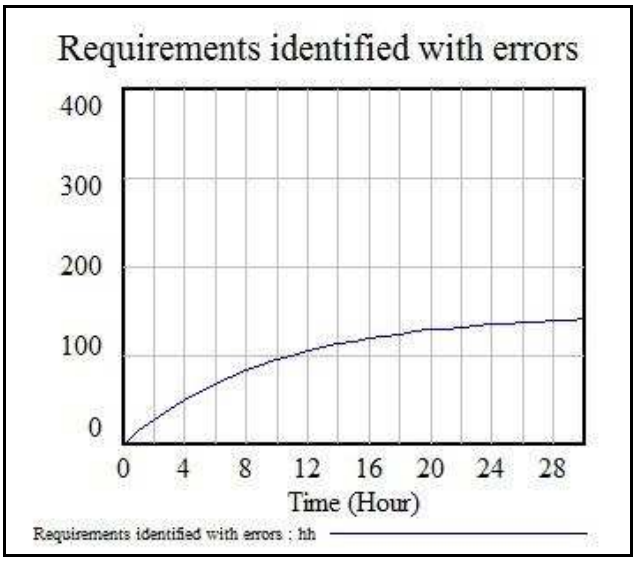

Figura 9. Resultado da simulação com a técnica Ad Hoc Reading.

Interpretando o gráfico da Fig. 9, percebe-se que a técnica utilizada encontrou 43% dos requisitos defeituosos esperados, mantendo a mesma configuração das simulações descritas anteriormente.

4.4 Checklist Based Reading

Na técnica de leitura baseada em checklists, já disseminada na indústria, o grupo de inspetores faz uso de uma mesma lista para a leitura e análise do artefato. Esta lista faz a relação entre os itens a serem verificados e os que devem ser entendidos como defeituosos. Cada tipo de artefato possui uma lista específica que pode ser: Documento de Requisitos, Casos de Uso, Cenários, Léxico Ampliado da Linguagem, Diagramas de Classes, Plano de Testes, Casos de Teste, Código, etc (SAYAO, 2012).

Essas listas são adaptáveis, de modo que os desenvolvedores podem levar em consideração os defeitos de maior ocorrência naquele ambiente. Eles poderão focar a atenção nos pontos que originam esses defeitos, e consequentemente haverá uma

diminuição no número de defeitos nos artefatos de software. Por outro lado, os defeitos que não forem caracterizados na checklist não serão detectados. Para diminuir esse problema, algumas regras básicas podem ser seguidas para a construção de uma lista mais eficaz: a) uma página deve ser suficiente para relacionar as questões a serem respondidas; b) as questões devem ser objetivas e precisas e c) os atributos de qualidade que devem estar presentes no artefato devem ser bem descritos, e as respostas às questões devem assegurar que tal atributo se encontra ou não presente (SAYAO, 2012).

Esta técnica de inspeção, quando aplicada a artefatos de requisitos, pode identificar os seguintes defeitos: sintaxe incorreta, inconsistência, ambiguidade, informação desnecessária, ausência de informação e requisitos não funcionais não explicitados (SAYAO, 2012).

O resultado da simulação para o cenário C6, com a utilização da técnica mencionada, é mostrado na Fig. 10.

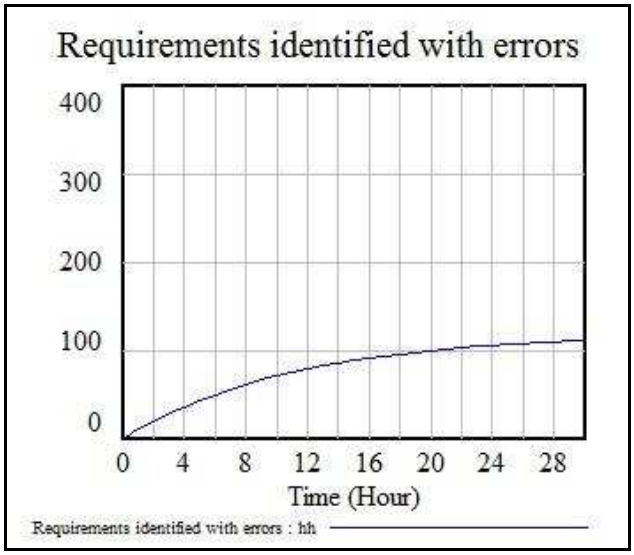

Figura 10. Resultado da simulação com a técnica Checklist Based Reading.

Analisando o gráfico com os resultados da simulação através da técnica Checklist Based Reading fica evidente que ela não é tão eficiente quanto às demais apresentadas. Foram identificados apenas 35% dos requisitos com problemas esperados.

4.5 Scenario Based Reading (Defect Based Reading)

Técnicas de Leitura Baseadas em Cenários utilizam a ideia de revisões ativas de projeto para atribuir responsabilidades específicas aos inspetores de uma equipe

de inspeção. Apesar dessas revisões ativas de projeto não mostrarem como a inspeção deve ser realizada, essas técnicas descrevem mais precisamente como procurar defeitos. O inspetor segue um conjunto de instruções e diretrizes, constituindo assim o chamado Cenário. O Cenário força o inspetor a fazer um papel de leitura ativo, exigindo, por exemplo, que ele escreva os casos de teste.

Uma das vantagens do uso de Cenários é que o inspetor adquire as reais características do sistema que está descrito no documento de requisitos comparado a inspetores que aplicam Checklists. Os inspetores devem exercer um papel mais ativo e pensam mais profundamente no sistema. Com um maior entendimento do sistema, os inspetores conseguem detectar defeitos menos visíveis. Os Cenários usam questões e checklists altamente especializados que ajudam a descobrir defeitos usando o trabalho já feito (BERTINI, 2006).

Dentro de uma mesma equipe de inspeção, todo inspetor utiliza um cenário diferente tendo consequentemente uma responsabilidade diferente. O objetivo da criação da Técnica de Leitura Baseada em Cenários é encontrar uma combinação de cenários que cubra todos os aspectos do documento a ser inspecionado e, ao mesmo tempo, seja o menos redundante possível (BERTINI, 2006).

O resultado da simulação para o cenário C7, com o uso da técnica descrita, é exibido na Fig. 11.

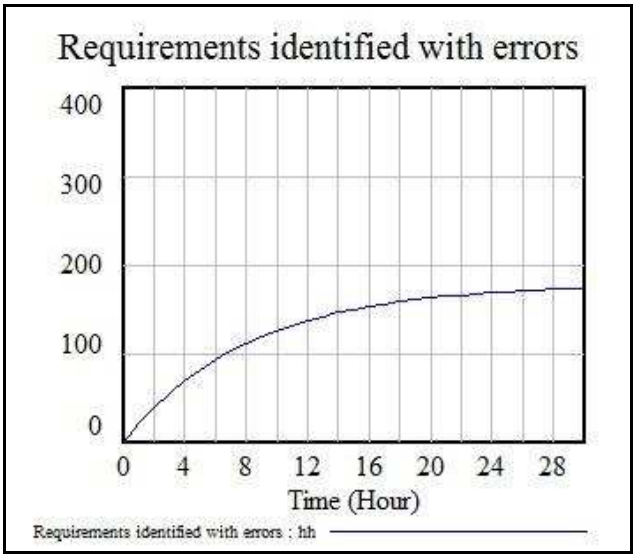

Figura 11. Resultado da simulação com a técnica Scenario Based Reading (Defect Based Reading).

Visualizando a Fig. 11, fica evidente que muitos requisitos com erros foram detectados por meio dessa técnica. De todos os requisitos com defeitos esperados, a Técnica de Leitura Baseada em Cenário conseguiu identificar 52%.

4.6 Perspective Based Reading

A razão básica da Leitura Baseada em Perspectivas (LBPe) é que diferentes consumidores ou stakeholders do documento a ser inspecionado estão interessados nos diferentes fatores de qualidade do documento. Correspondentemente, LBPe define perspectivas dos stakeholders para o documento de requisitos. Essas perspectivas foram desenvolvidas de um modo que os inspetores são estimulados a enxergar o ponto de vista do usuário, do projetista e do testador do sistema, utilizando a perspectiva correspondente (BERTINI, 2006).

A técnica LBPe divide-se em três partes principais: introdução, instruções e perguntas. A introdução é um resumo que explica como o inspetor deverá usar a perspectiva. As instruções informam ao inspetor como ele deve extrair a informação do documento de requisitos. O inspetor deve construir um modelo físico do sistema. Os efeitos destas instruções são três: a) as instruções ajudam os inspetores a decompor o documento de requisitos em partes menores, pois deste modo podem ser gerenciados de forma mais fácil; b) os inspetores são forçados a trabalhar ativamente com os requisitos e com a perspectiva; c) focalizam a atenção do inspetor nas

informações mais relevantes. Os inspetores ganham um entendimento mais significativo do sistema e, além disso, asseguram que eles estão mais bem preparados para as próximas atividades, como a reunião para coletar os defeitos (BERTINI, 2006).

Existem várias versões de LBPe. Com essa técnica é possível definir até três perspectivas: a) a perspectiva do usuário que exige que o inspetor desenvolva casos de uso para o sistema; b) a perspectiva do projetista que exige que o inspetor construa um diagrama de fluxo de dados do sistema; e c) a perspectiva do testador que exige que o inspetor realize casos de teste (BERTINI, 2006).

Pode-se classificar LBPe como sistemática (porque uma perspectiva orienta o inspetor com respeito a como e onde procurar defeitos), específica (porque o inspetor concentra-se em certos aspectos) e distinta (porque os inspetores com a equipe de inspeção usam perspectivas diferentes) (BERTINI, 2006).

O resultado da simulação para o cenário C8, com o uso da técnica descrita, é apresentado na Fig. 12.

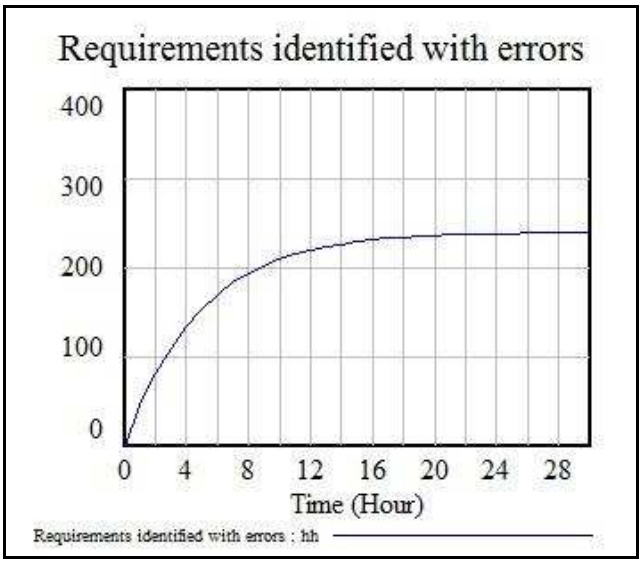

Figura 12. Resultado da simulação com a técnica Perspective Based Reading.

Analisando o gráfico da Fig. 12, percebe-se uma melhora no número de requisitos encontrados com defeitos em relação aos gráficos anteriores. Com a utilização da técnica Perspective Based Reading foi possível detectar 70% dos requisitos esperados com algum tipo de erro.

5 COMPARAÇÃO DOS RESULTADOS DOS CENÁRIOS

É importante ressaltar que as simulações foram executadas considerando o uso da técnica de entrevista para a elicitação de 1000 requisitos durante um período de 30 horas de trabalho de um elicitador engenheiro/analista e de um stakeholder experiente. Portanto, não foi considerado o número total de requisitos do sistema.

A Fig. 13 mostra uma comparação entre os cenários com base no número de requisitos com erros detectados para cada tipo de técnica utilizada nos cenários simulados.

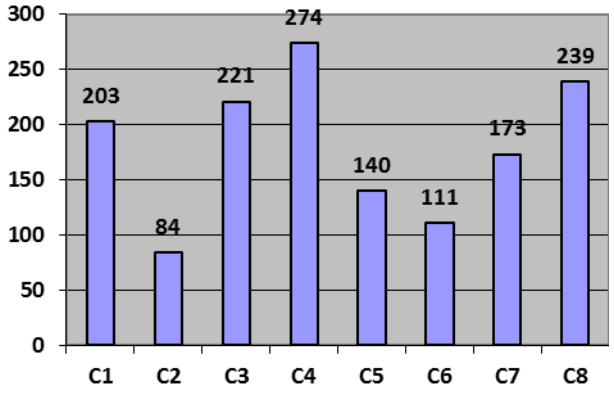

Figura 13. Gráfico de comparação entre os cenários

Analisando-se o gráfico da Fig. 13, percebe-se que a técnica N-fold Inspection com N = 10 foi a que obteve melhores resultados em relação ao número de requisitos com defeitos encontrados. Porém, esta técnica tem um alto custo e exige um enorme esforço, tornando-a pouco atrativa para o uso. Depois desta técnica, a que obteve melhores resultados foi a técnica Perspective Based Reading que permitiu encontrar 239 requisitos com erros. A técnica N-fold Inspection com N = 1 foi a que teve o pior desempenho, identificando apenas 84 requisitos com algum problema. As técnicas Formal Specification Method, Ad Hoc Reading, Checklist Based Reading e Scenario Based Reading (Defect Based Reading), identificaram 203, 140, 111 e 173 requisitos com erros, respectivamente; números que representam resultados bastante significativos.

6 CONCLUSÕES E TRABALHOS FUTUROS

O modelo de dinâmica de sistemas apresentado permite verificar, por meio de simulações, os efeitos do uso de diferentes técnicas de detecção de erros em

requisitos. Esse modelo pode ser utilizado como uma ferramenta de apoio à tomada de decisão em projetos de software, permitindo comparar a eficácia e o comportamento de cada técnica antes de aplicá-la em um projeto real.

O modelo permite analisar, por exemplo, o quanto uma técnica de identificação de erros em requisitos pode ser útil em um projeto. Isso permite verificar qual técnica é mais adequada para um determinado projeto, já que todas elas possuem custos e exigem esforços diferentes.

Com o intuito de desenvolver um modelo capaz de produzir resultados cada vez mais próximos da realidade, os próximos passos desse trabalho são: construir um modelo de dinâmica de sistemas que contextualize diferentes técnicas de elicitação de requisitos e de detecção de erros em requisitos; calibrar e validar os modelos com dados disponíveis na literatura e com dados obtidos das empresas parceiras do Laboratório de Engenharia de Software (LES) da Universidade Federal de Viçosa; e integrar os modelos do presente trabalho com os modelos produzidos por Ambrósio (2008) e Hermsdorf (2011) para obter um modelo mais completo e robusto, abordando várias atividades da fase de requisitos em processos de desenvolvimento de software.

Além disso, para facilitar o acesso das empresas ao modelo, será permitido que simulações e análises sejam efetuadas remotamente, via Web.

AGRADECIMENTOS

A CAPES, Sydle, Gapso, CNPq pelo apoio financeiro.

2.2 Simulations using system dynamics models to select inspection techniques in requirements elicitation

Jailton Jr. De S. Coelho, José Luis Braga, Bernardo Giori Ambrósio

Keywords: Inspections; requirements errors; system dynamics.

1 INTRODUCTION

Requirements specification is the basis for good results in applying software engineering techniques and models for software development. Having in hands a requirements specification artifact, it is possible to detect and correct requirements errors, thus avoiding less rework in later phases of the process. Inspection techniques deliver better results in early requirements error detection. They are very effective and lead to the detection of inconsistent and incorrect requirements specification.

This work describes an ongoing research project, aiming at making available a system dynamics model to be used in decision making on inspection techniques choice. System analysts and designers should be able to forecast the adequacy of each technique before applying them in a real problem, by running simulations using our model.

System dynamics is a technique well suited to model and run simulations on this problem, involving variables that have dynamic behavior and variability over time. System dynamics models allow the systemic understanding of problems, making visible and clear decision variables and their connections (AMBROSIO, 2008).

There are some previously related works available in the literature related to developing system dynamics models focused on the requirements extraction and specification discipline of software development processes. The ones most related to this work are (AMBROSIO, 2008) and (HERMSDORF, 2011). This work compares the use of Interviewing (HERMSDORF, 2011), Laddering (HERMSDORF, 2011), Sorting (HERMSDORF, 2011) and Protocol Analysis (HERMSDORF, 2011) requirements elicitation techniques associated with: Formal Specification Method (AHMED, 2009), N-fold Inspection (AHMED, 2009), Ad Hoc Reading (AHMED, 2009), Checklist Based Reading (AHMED, 2009), Scenario Based Reading (Defect

Based Reading) (AHMED, 2009), and Perspective Based Reading (AHMED, 2009) inspection techniques.

2 MODELING THE USE OF INSPECTION TECHNIQUES ASSOCIATED WITH REQUIREMENTS ELICITATION TECHNIQUES

We followed the steps of a method proposed by Ambrósio (2008) to obtain our model. It consists in three main steps that are divided in one or more steps each. First step consists of getting a first raw model relating the problem main variables using influence diagrams notation (AMBROSIO, 2008). In the second step, influence diagrams are mapped into stock-flow system dynamics diagrams, and the relationships between variables, to be used on running simulations, are selected. In the third step, models built so far are refined by running experiments and tuning them incrementally, until the models become stable and deliver results compatible with current practice in the area.

2.1 System Dynamics model

The model presented in Figure 1 was obtained by applying the method steps introduced last section. We first built a influence diagram model and then transformed it by conveniently mapping its elements into stocks, flows and converters.

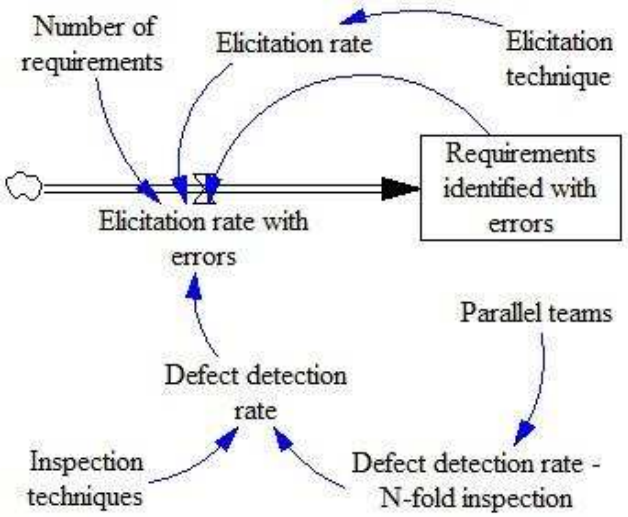

Figure 1. System dynamics stock-flow model used in simulations

In Figure 1, there is only one stock, “Requirements identified with errors”, that is our main goal. The model has also only one input flow, “Elicitation rate with

errors". The other elements are external or feedback variables that should be considered in simulations. The flow "Elicitation rate with errors" depends on "Elicitation rate", "Number of requirements", "Defect detection rate" and "Requirements identified with errors". "Number of requirements" represents the number of requirements set up for simulations in each run by using a specific "Elicitation technique". "Elicitation rate" refers to the estimated number of requirements elicited per hour. "Parallel teams" represents the number of teams working in parallel while using the N-fold technique, represented by "Defect detection rate – N-fold inspection". Inspection techniques that can be simulated are represented by "Inspection techniques".

2.2 Control Panel

The Control Panel is a powerful resource offered by the tools that support system dynamics model construction and simulations. It allows the adjustment of decision variables values before running simulations, thus allowing comparing results with different scenarios. It makes easy the interaction of decision makers with the model, and is conceptually very simple.

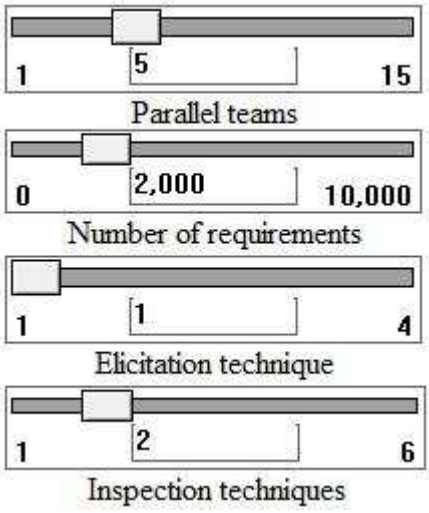

Figure 2. Control Panel to setup simulation parameters

3 SIMULATIONS AND RESULT ANALYSIS

We designed eight scenarios for running simulations involving the six inspection techniques selected in this work. In all of the scenarios, we considered that two elicitors (one engineer/analyst and one student), six experienced stakeholders

and two inexperienced stakeholders work together in eliciting 2000 requirements in 30 work/hours, using the Interviewing technique. These values were taken from the work of Hermsdorf (2011) that also sets the occurrence of 593 requirements with errors among those 2000 elicited. In that work, it was also concluded that the Interviewing technique is the one that delivers the best results in requirements elicitation.

Scenarios C1 through C8 use the parameters and assumptions described in the last paragraph, but they differ regarding the inspection technique adopted. In C1, it was assumed that Formal Specification Method would be used; in C2, C3 and C4 it was assumed N-fold Inspection with 1, 5 and 10 parallel teams, respectively; and C5, C6, C7 e C8 were assumed to use the techniques Ad Hoc Reading, Checklist Based Reading, Scenario Based Reading (Defect Based Reading) and Perspective Based Reading, respectively. The data and relationship equations we used in our model were taken from the literature.

4 COMPARING SIMULATION RESULTS

Figure 3 is a graph that shows the results of simulations run on the eight scenarios. The technique N-fold Inspection with 10 parallel teams was the one that delivered best results, leading to the identification of 479 requirements with errors. But this technique has a higher cost of application and demands higher efforts to be applied, making it less attractive. The second best result was obtained by using Perspective Based Reading leading to the identification of 418 requirements with errors. N-fold Inspection taking one team delivered the worst result in our scenarios, leading to only 146 requirements with errors. Formal Specification Method, Ad Hoc Reading, Checklist Based Reading and Scenario Based Reading (Defect Based Reading), leaded to respectively 355, 246, 195 and 304 requirements with problems.

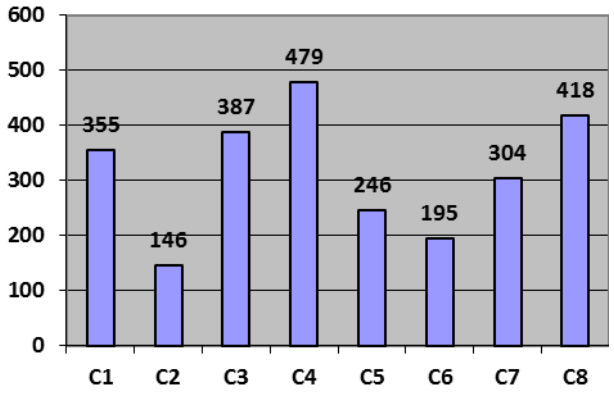

Figure 3. Graph showing simulation results

5 CONCLUSION

The model we introduced in this paper can be used in real simulations, allowing Project Managers to make more informed and safer decisions on techniques choice. It can be easily used as a decision making tool support in managing software projects.

ACKNOWLEDGEMENTS

We thank CAPES, Sydle, Gapso, and CNPq for financial support.

2.3 Modelo de Dinâmica de Sistemas para Apoio a Decisões no Processo de Inspeção de Software

Jailton Jr. De S. Coelho[1] , José Luis Braga[1]

RESUMO

Uma das formas de melhorar a qualidade de software é a aplicação de inspeções nos artefatos desenvolvidos ao longo do processo de desenvolvimento. Embora as vantagens de se utilizar inspeções de software sejam conhecidas, a forma como as inspeções são realizadas na prática pelas organizações de software ainda é pouco sistematizada e seu potencial raramente é explorado. Este artigo apresenta um modelo de dinâmica de sistemas, envolvendo grande parte das variáveis que influenciam fortemente na eficiência da inspeção. Através do modelo é possível reproduzir cenários nos quais seria caro ou perigoso experimentar na realidade, sendo possível analisar os impactos que a inspeção pode trazer no processo de desenvolvimento, diminuindo riscos.

Keywords: inspeção de software, dinâmica de sistemas, detecção de defeitos.

1 INTRODUÇÃO

A tomada de decisão é uma atividade importante e complexa na engenharia de software. Geralmente é um processo não sistemático, uma vez que tipicamente se baseia em experiência pessoal sem o uso de modelos explícitos. O desenvolvimento de software é um processo onde cada pessoa envolvida constantemente toma decisões, tanto técnicas quanto gerenciais.

A capacidade de enxergar e compreender o sistema como um todo é fundamental nas decisões. Em projetos de software, os gestores, quando buscam visualizar de forma integrada todos os fatores de influência do seu processo de desenvolvimento, terão maior capacidade de tomar decisões mais adequadas. Entretanto, gestores acabam tomando decisões reativas, considerando apenas o problema presente, sem relacioná-lo com o seu ambiente e demais fatores correlacionados. Isso se torna um risco relevante no projeto, já que decisões

gerenciais tomadas de maneira equivocada podem levar a grandes prejuízos e, mais do que isso, podem causar falhas críticas no software. Logo, é indispensável o uso de ferramentas que auxiliem os gerentes em suas tomadas de decisão, identificando os riscos antes de eles se materializarem (HERMSDORF, 2011).

A qualidade de um produto de software é um aspecto que deve ser acompanhado durante todo o processo de desenvolvimento. A participação humana é um fator de grande presença nos projetos de software, justificando a ocorrência de defeitos, até mesmo se as melhores técnicas e procedimentos forem utilizados. Com isso, fica clara a importância de um processo de garantia da qualidade que atue em todas as fases do processo de desenvolvimento (BARTIÉ, 2002). Revisões de artefatos de software têm se mostrado uma abordagem eficiente e de baixo custo para encontrar defeitos, reduzindo o retrabalho e melhorando a qualidade dos produtos (KALINOWSKI, 2004).

A inspeção é um tipo particular de revisão que contribui para garantir a qualidade do produto de software. Todas as etapas do processo de desenvolvimento de software são suscetíveis à incorporação de defeitos, que podem ser detectados pela inspeção e posteriormente removidos.

É importante destacar que quanto mais cedo esses defeitos forem removidos, menor será o custo de desenvolvimento e manutenção do produto. Experiências têm comprovado que a inspeção, quando realizada no início do processo de desenvolvimento do software, pode levar à detecção de 60% a 90% dos defeitos potenciais (BOEHM, 2001).

Segundo (CIOLKOWSKI, 2003), embora muitas organizações de software realizem revisões, a forma como as revisões são realizadas ainda é pouco sistematizada, pouco conhecimento da área de inspeções de software é utilizado e assim seu potencial raramente é explorado. Este argumento é baseado em três observações sobre as organizações participantes do levantamento de dados apresentado no artigo: (1) revisões raramente cobrem sistematicamente todas as fases de desenvolvimento de software (40% realizam inspeções em requisitos e 30% realizam inspeções em código), (2) muitas vezes a atividade de detecção de defeitos não é realizada sistematicamente (cerca de 60% não conduzem uma atividade de detecção de defeitos regularmente), e (3) cerca de 40% das organizações não coletam dados e 18% coletam dados e não os utilizam em suas análises (KALINOWSKI, 2004).

Dada a importância do processo de inspeção e a complexidade das relações entre as variáveis dinâmicas envolvidas nessa fase do processo de desenvolvimento de software, torna-se indispensável o uso de ferramentas que permitam enxergar o contexto como um todo (AMBRÓSIO, 2011). A dinâmica de sistemas (MADACHY, 2007) é uma dessas ferramentas, e será explorada neste trabalho para permitir uma melhor visualização do contexto e auxiliar na tomada de decisões sistêmicas (HERMSDORF, 2011).

A utilização da dinâmica de sistemas no contexto da engenharia de software busca contribuir para a simulação de ambientes de desenvolvimento de software, servindo como um modelo de apoio à decisão para os gestores, possibilitando assim uma melhoria de qualidade em todo o processo de desenvolvimento de software (DAIBERT, 2010).

Com o intuito de apoiar decisões sob este ponto de vista, este artigo apresenta um modelo de dinâmica de sistemas, que permite avaliar os impactos da utilização de inspeções em várias etapas do ciclo de desenvolvimento de projetos de software, economizando recursos e permitindo reproduzir cenários nos quais seria caro ou perigoso experimentar na realidade. Através do modelo, gerentes de projetos podem configurar e simular cenários para auxiliá-los na análise e tomadas de decisão relativas às características da empresa, permitindo testar situações novas, antes de colocá-las em prática. O modelo foi construído embasado em dados disponíveis na literatura, garantindo sua consistência.

Variáveis como o número e experiência dos inspetores, técnica de leitura, tipo e tamanho do documento, entre outras, influenciam diretamente na qualidade do processo de inspeção. Não gerenciar estas variáveis no processo de inspeção contribui para a ineficiência do processo, uma vez que elas influenciam fortemente na sua qualidade (DAIBERT, 2010).

2 INSPEÇÃO DE SOFTWARE

A inspeção de software é uma técnica que vem sendo adotada cada vez mais no processo de desenvolvimento, para encontrar e reparar defeitos em documentos de requisitos, design, código e testes (KELLY, 1992). Uma das principais razões para essa crescente adoção é que ela pode ser usada nas fases iniciais do ciclo de

desenvolvimento de software, detectando defeitos, reduzindo os custos e aumentando a qualidade do software (FRANZ, 1994).

A aplicação de inspeções durante o desenvolvimento do projeto de forma bem planejada pode trazer diversos benefícios (BASILI, 1996):

- Aprendizado. Inspetores experientes podem identificar padrões de como os defeitos ocorrem e definir diretrizes que auxiliem na detecção dos mesmos. Além disto, bons padrões de desenvolvimento podem ser observados durante a inspeção, sendo possível sua descrição como melhores práticas para a organização.
- Integração entre processos de detecção e de prevenção de defeitos. Saber onde e quando os defeitos ocorrem auxilia no estabelecimento de planos de contingência que evitem a sua ocorrência.
- Produtos mais inteligíveis. Os autores dos artefatos de software, sabendo que estes serão inspecionados, passarão a produzir artefatos de uma forma que sua compreensão seja facilitada. A produção de artefatos mais inteligíveis, além de facilitar a inspeção, trará benefícios para as fases seguintes do processo de desenvolvimento, incluindo principalmente a fase de manutenção.
- Dados defeituosos ajudam a melhorar o processo de desenvolvimento do software. Analisando a causa dos defeitos encontrados é possível fazer ajustes no processo para evitar a ocorrência de defeitos deste mesmo tipo.

O processo básico da inspeção consiste de seis etapas consecutivas (KELLY, 1992):

- Planejamento. Desempenhado por um moderador cuja função é definir o contexto da inspeção (descrição da inspeção, técnica de leitura a ser utilizada na detecção de defeitos, documento a ser inspecionado, autor do documento, dentre outros), selecionar os inspetores e distribuir o material a ser inspecionado.
- Detecção de Defeitos. Os inspetores selecionados têm a tarefa de buscar defeitos no documento entregue a eles e, ao longo da inspeção, produzir uma lista dos possíveis defeitos.

- Coleção de Defeitos. O moderador tem a função de juntar as listas de defeitos dos inspetores, eliminando os defeitos repetidos (encontrados por mais de um inspetor), mantendo só um registro para cada defeito.
- Discriminação de Defeitos. O moderador juntamente com o autor do documento e os inspetores discutem os defeitos a fim de classificá-los como defeito de fato ou como falso positivo. Os falso positivos são então eliminados da lista de defeitos, permanecendo apenas os defeitos reais.
- Retrabalho. Os defeitos que fazem parte da lista são corrigidos pelo autor do documento, produzindo no final um relatório de correção dos defeitos.
- Continuação. É decidido pelo moderador se uma nova inspeção deve ou não ocorrer.

Existem dois tipos básicos de inspeção de software, a inspeção de documentos de requisitos e a inspeção de código-fonte. Na inspeção em documentos de requisitos, os inspetores têm a tarefa de analisar os documentos com o objetivo de encontrar defeitos, aproveitando o momento que eles são mais fáceis e baratos de serem corrigidos. Já na inspeção de código-fonte, os inspetores realizam uma análise estática do código, visando a encontrar erros. Esse tipo de inspeção pode diminuir a complexidade dos programas, pois os subprogramas são feitos em um estilo consistente, obedecendo a padrões pré-estabelecidos (DUNSMORE, 2000).

Podem ser detectados os seguintes defeitos quando se realiza inspeções em documentos de requisitos: Defeito de omissão: esse defeito ocorre quando informações necessárias do sistema são omitidas, como por exemplo, a omissão de uma funcionalidade ou qualidade do sistema; Defeito de fato incorreto: este defeito é caracterizado quando informações nos artefatos do sistema são contraditórios com o domínio da aplicação; Defeito de inconsistência: este defeito está relacionado a informações que aparecem mais de uma vez no artefato, só que de forma diferente em cada ocorrência, causando incoerência; Defeito de ambiguidade: esse defeito ocorre quando a informação transmitida leva a múltiplas interpretações; Defeito de informação estranha: esse defeito é classificado quando é encontrada uma informação relacionada ao domínio, entretanto ela não é necessária para o sistema em questão (DUNSMORE, 2000).

Inspeções realizadas em código-fonte podem encontrar os seguintes defeitos: Defeitos de Omissão: esse defeito ocorre quando há omissão de algum elemento no

programa, como uma função que converte um valor em outro; Defeitos de Comissão: esse defeito é caracterizado por um segmento de código incorreto, quando, por exemplo, uma expressão calcula algum valor de forma incorreta; Defeito de inicialização: esse defeito é classificado quando uma estrutura de dados é inicializada de forma incorreta; Defeitos de computação: esse defeito ocorre quando a geração do valor de uma variável é computada de maneira errada; Defeito de controle: esse defeito acontece quando um valor de entrada é direcionado para um caminho de controle falso; Defeito de interface: esse defeito é classificado quando um módulo do programa realiza suposições sobre dados, sendo que esses dados não fazem parte do escopo do projeto; Defeitos de dados: esse defeito ocorre quando uma estrutura de dados é utilizada de maneira equivocada; Defeitos de cosmética: esse defeito é caracterizado por erros de ortografia e gramática no programa (DUNSMORE, 2000).

Equipes de inspetores podem ser compostas pelos seguintes papéis, com suas respectivas funções (LAITENBERGER, 2000):

- Organizador: Planejar todas as atividades de inspeção que fazem parte do projeto.
- Moderador: Conduzir as atividades e controlar as reuniões.
- Autor: Criar o produto de trabalho a ser inspecionado e corrigir os defeitos identificados.
- Apresentador: Descrever as seções do produto de trabalho para os inspetores, explicando e interpretando.
- Redator: Classificar e registrar todos os defeitos encontrados durante a reunião de inspeção.
- Coletor: Coletar os defeitos encontrados pelos inspetores caso não seja realizada mais nenhuma reunião de inspeção.
- Inspetor: Encontrar erros no documento. Todos os participantes podem atuar como inspetores em adição a quaisquer outras responsabilidades.

A inspeção utiliza-se da revisão baseada na leitura e entendimento de um artefato de software a fim de encontrar defeitos. Entretanto, muitos desenvolvedores possuem facilidades para escrever documento de requisitos e código de software, mas não é oferecido a eles nenhum suporte relacionado à leitura adequada dos mesmos. Uma das soluções é então fazer uso de técnicas de leitura. Técnicas de leitura podem ser definidas como uma série de passos ou procedimentos cuja

finalidade é permitir que o inspetor adquira conhecimento profundo do produto de software a ser inspecionado, com o objetivo de detectar defeitos (FAGAN, 1986).

Técnicas de leitura contribuem para o aumento da compreensão sobre algum artefato de software. Esta compreensão deve ser suficiente a ponto de permitir que os inspetores identifiquem tanto as informações importantes para a execução de uma determinada tarefa, como a relação destas informações com o problema de que se está tratando (FAGAN, 1986).

Algumas das principais técnicas de leitura são (LAITENBERGER, 2000):

- Ad-hoc: a técnica de leitura Ad hoc não oferece nenhum suporte técnico de como detectar defeitos em um artefato de software. Neste caso, a detecção de defeitos depende totalmente da habilidade, do conhecimento e da experiência de um inspetor.
- Checklist (LBCh): a técnica de leitura Checklist é baseada em uma série de perguntas (frequentemente questões de sim/não) sobre assuntos do documento a ser inspecionado. As questões são elaboradas de acordo com o tipo de documento de software que se pretende inspecionar, e muitas vezes não é dada nenhuma instrução concreta sobre como usar uma lista de verificação.
- Cenário (LBCe): a essência da ideia de leitura baseada em Cenário é o uso da noção de cenários que fornecem orientação personalizada para inspetores sobre como detectar defeitos. Um cenário pode ser um conjunto de perguntas ou uma descrição mais detalhada para um inspetor sobre a forma de realizar a análise do documento. Seguindo um cenário, o inspetor adquire conhecimento sobre a essência do sistema que está descrito, forçando-o a exercer um papel mais ativo e a pensar mais profundamente no sistema, achando defeitos mais sutis.
- Stepwise Abstraction (SA): é uma técnica que fornece instruções de leitura mais estruturadas e precisa para os documentos de código. Um inspetor lê uma sequência de declarações no código e abstrai uma função de calcular essas declarações. Este procedimento é repetido até que a última função do

artefato de código inspecionado seja captada, comparando no final essas funções com a especificação.

- Análise de Pontos de Função (APF): a abordagem baseada em Análise de Pontos de Função (APF) define um sistema de software em termos de suas entradas, arquivos, consultas e saída. Os cenários de pontos de função são desenvolvidos em torno desses itens. Um cenário de Pontos de Função consiste em perguntas e direciona o foco de um inspetor para um item de ponto de função específico dentro do documento de requisitos inspecionados. A principal ideia por trás dessa técnica é que os inspetores se concentram em diferentes classes de defeitos enquanto examinam um documento de requisitos. Para cada classe de defeitos, existe um cenário constituído por um conjunto de perguntas que um inspetor tem de responder durante a leitura. As perguntas ajuda o inspetor a detectar defeitos de uma classe específica.
- Perspectiva (LBPe): a ideia principal por trás da técnica de leitura baseada em perspectiva é a de que um produto de software deve ser inspecionado a partir da perspectiva dos diferentes stakeholders. As perspectivas desta técnica são elaboradas de modo que os inspetores são forçados a enxergar o ponto de vista do usuário, do projetista e do testador do sistema, utilizando a perspectiva correspondente. A técnica LBPe divide-se em três partes principais: introdução, instruções e perguntas. A introdução descreve como o inspetor irá utilizar a perspectiva. As instruções informam ao inspetor como ele deve extrair a informação do documento de requisitos. E as perguntas representam um questionário altamente especializado, que focaliza a atenção do inspetor para aspectos específicos da sua tarefa, ajudando a detectar os defeitos.
- N-fold: o método de leitura N-fold consiste na replicação do processo de realização de inspeções formais para detecção de defeitos usando N equipes trabalhando em paralelo com um único moderador que é o responsável por coordenar e reunir os resultados. Essa técnica é aplicada considerando a hipótese de que a existência de equipes diferentes melhora a eficiência do processo, detectando defeitos que não seria encontrado por uma única equipe de inspetores.

3 DINÂMICA DE SISTEMAS

A disciplina Dinâmica de Sistemas foi desenvolvida na década de 50 pelo engenheiro eletricista Jay Forrester, publicando em 1961 o livro "Industrial Dynamics" (Dinâmica Industrial). Mas só por volta de 1968 é que a disciplina começou a ganhar força nos famosos modelos de estudos estratégicos urbanos e mundiais, divulgados e disseminados pelos livros: "Urban Dynamics" (Dinâmica Urbana) e "World Dynamics" (Dinâmica do Mundo) (FORRESTER, 1980).

A dinâmica de sistemas é uma técnica descritiva, que pode ser utilizada na modelagem e simulação de sistemas, fortemente baseada no pensamento e análise sistêmicos e na teoria matemática dos sistemas dinâmicos, permitindo levar em consideração as variações dinâmicas dos problemas.

Sua enorme utilidade permite a compreensão de como as políticas adotadas, ou a própria estrutura do sistema, afetam ou determinam o seu comportamento dinâmico, contribuindo na previsão de situações futuras e antecipando colapsos. O uso dessa técnica permite a análise e compreensão de problemas e situações de maneira integrada e interconectada, deixando claras as relações existentes entre as variáveis de decisão (AMBRÓSIO, 2011).

A dinâmica de sistemas é um método para modelar e analisar o comportamento de sistemas complexos, formados por diversas variáveis que se relacionam de forma não linear. Nesses sistemas, as diversas variáveis presentes exercem influência umas sobre as outras, formando laços de realimentação e determinando o comportamento do sistema. Esses laços são críticos, pois alterações em algumas variáveis podem desencadear comportamentos que podem levar todo o sistema a entrar em colapso. A dinâmica de sistemas permite expressar as influências entre as variáveis e capturar os laços de realimentação responsáveis pelo comportamento do sistema (HERMSDORF, 2011).

Diante da necessidade de realizar alguma intervenção no sistema para resolver algum problema, é fundamental conhecer as verdadeiras causas do comportamento indesejado para que a intervenção não seja apenas uma solução paliativa e agrave ainda mais o problema. A modelagem com dinâmica de sistemas facilita a descoberta das causas do problema e também, por meio de simulações utilizando o modelo, permite analisar os impactos e os efeitos colaterais das alterações planejadas antes que elas sejam implementadas no sistema real (FORRESTER, 1980).

Um modelo de dinâmica de sistemas possui dois elementos principais: os estoques, que são os recursos acumuláveis do sistema, e os fluxos, que são funções que representam as decisões ou políticas da empresa com relação aos usos e acúmulos dos estoques ou recursos. Há também os conversores, ou variáveis simples, que são os elementos do modelo que exercem influência sobre os valores dos fluxos responsáveis pela variação dos estoques (FORRESTER, 1980).

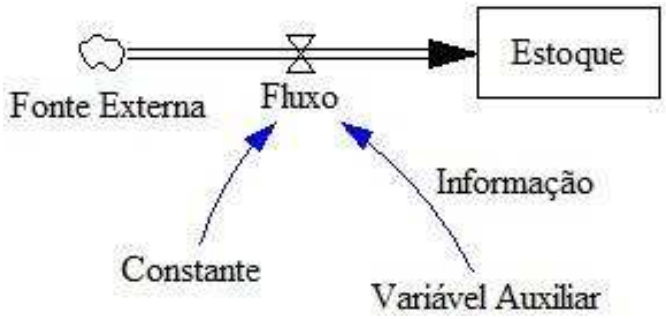

Figura 1. Elementos básicos de um modelo genérico de estoque e fluxo (MADACHY, 1996).

Segue a descrição dos elementos da Fig. 1:

- Estoques: representam os acúmulos ou decrementos ao longo do tempo. Seus valores são influenciados através dos fluxos de entrada e/ou de saída.
- Fonte externa: são recursos ou repositórios infinitos utilizados pelos fluxos que não são especificados no modelo.
- Fluxos: podem ser consideradas as "ações" no sistema. Eles influenciam diretamente nos estoques e podem representar as decisões e políticas adotadas.
- Variável auxiliar: são os conversores com o objetivo de ajudar a elaborar detalhes da estrutura de representação do modelo estoque e fluxo. Eles se relacionam com os fluxos através de setas unidirecionais.
- Informação (setas unidirecionais): são utilizadas para representar fluxos de informação. Estas setas podem ajudar na construção de loops de realimentação entre os elementos. São responsáveis por interligar as variáveis do modelo.
- Constantes: representam as variáveis que não se alteram durante toda simulação.

Em todo modelo estoque e fluxo da dinâmica de sistemas, há um sistema de equações matemáticas que pode ser utilizado pelo software de simulação. Através

desse sistema de equações é possível avaliar a dinâmica do sistema que se pretende modelar, permitindo que a simulação do sistema seja feita de forma manual (AMBRÓSIO, 2011). A dinâmica de sistemas não se associa de forma direta a nenhuma disciplina ou área de conhecimento específica, ou seja, ela pode ser usada para modelar sistemas com comportamento dinâmico de qualquer domínio de conhecimento. A dinâmica de sistemas faz uso de um vocabulário comum, diminuindo as dificuldades de comunicação entre os especialistas envolvidos na construção dos modelos (HERMSDORF, 2011).

4 MODELO DE DINÂMICA DE SISTEMAS PARA OTIMIZAÇÃO DO PROCESSO DE INSPEÇÃO DE SOFTWARE

O modelo de dinâmica de sistemas apresentado neste trabalho foi construído seguindo um método proposto por (AMBRÓSIO, 2011), constituído por três etapas subdivididas em um ou mais passos. Na primeira etapa, são identificadas as variáveis envolvidas no processo ou problema que se pretende modelar. Em seguida, identificam-se os relacionamentos existentes entre essas variáveis e por último, diagramas de influência são construídos para facilitar a visualização das relações entre as variáveis. Na segunda etapa os diagramas de influência são mapeados em modelos de dinâmica de sistemas. Na terceira e última etapa, os modelos são refinados e os relacionamentos entre seus elementos são quantificados repetitivamente, até que seja obtido um modelo finalizado e validado

As variáveis presentes no modelo fazem parte dos resultados de uma pesquisa realizada por (LAITENBERGER, 2000). Seus estudos apontaram que os fatores que causavam maiores impactos na quantidade de defeitos detectados com o uso da inspeção são:

- Características do time: as características do time referem-se ao número de pessoas envolvidas no processo e a experiência de cada uma delas.
- Esforço: o esforço gasto com inspeções é influenciado fortemente pelas características do time e do produto de software.
- Técnicas de leitura: as técnicas de leitura descrevem como o revisor deve ler o artefato para encontrar defeitos. Elas ajudam os inspetores no processo de detecção de defeitos provendo uma sequência bem definida de passos para a leitura dos artefatos.

- Organização da atividade de detecção de defeito: a organização da atividade de detecção de defeitos refere-se ao número de inspeções que será realizada e ao número de inspetores que farão parte da equipe.
- Características do produto: as características do produto são representadas pelo tipo, complexidade, tamanho e qualidade inicial do produto que deseja inspecionar. A qualidade do produto se refere ao número de defeitos que podem estar contidos no artefato de software. O ciclo de vida se refere à fase do desenvolvimento de software em que será empregada a inspeção. Podem ser realizadas inspeções em vários de tipos de documentos de software, são eles: Requisitos de Softwares, Design, Design Detalhado, Código-fonte e Testes.

Diagramas de influência têm como principal objetivo simplificar a visualização das relações de causa e efeito e realimentação de informações de um sistema (AMBRÓSIO, 2011). Sua notação baseia-se em diagramas simples com palavras e setas, não possui relação alguma com a notação utilizada na dinâmica de sistemas, porém esses diagramas servem como base para a obtenção de um modelo (HERMSDORF, 2011).

Após ser construído o diagrama de influência, foi realizado o mapeamento do mesmo em um modelo estoque e fluxo da dinâmica de sistemas. Para fazer essa transformação foi utilizada a ferramenta Vensim (VENSIM, 2010), em sua versão gratuita para uso acadêmico, que possibilita a criação dos elementos da dinâmica de sistemas, variáveis auxiliares, constantes, estoques e fluxos. Para se fazer essa conversão foram utilizadas as regras descritas a seguir:

- Variáveis cujo valor representa acúmulos ou decrementos ao longo das iterações da simulação, são transformadas em estoques.
- Variáveis que influenciam diretamente um estoque são mapeadas em fluxos ou taxas.
- Variáveis cujos valores representam os parâmetros de entrada do modelo, são convertidas em constantes.
- Variáveis não classificadas em nenhuma categoria anterior (estoque, taxa e constante) são transformadas em variáveis auxiliares.

O resultado do mapeamento do diagrama de influência construído inicialmente para o problema em um modelo estoque e fluxo é apresentado na Fig. 2, resultando em 3 estoques, 8 constantes, 9 variáveis auxiliares e 2 taxas.

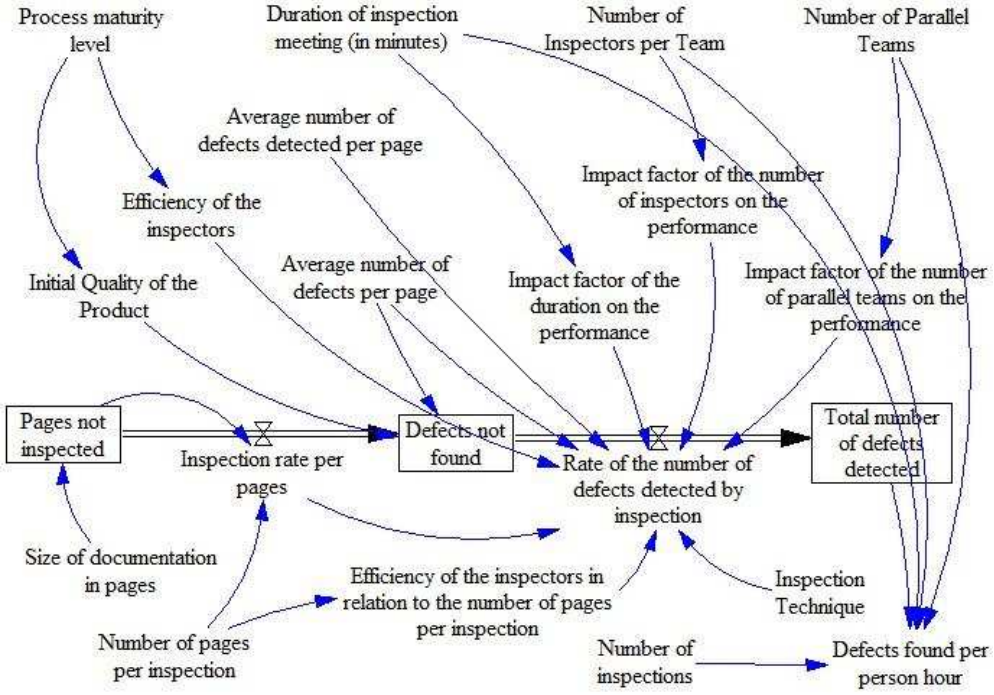

Figura 2. Modelo estoque e fluxo da Dinâmica de Sistemas.

A variável Rate of the number of defects detected by inspection representa o número de defeitos que são encontrados por uma única inspeção simulada. Ela é a variável mais importante do modelo, por ser influenciada pela maioria das variáveis e também por ser o fator determinante no número total de defeitos detectados (Total number of defects detected) com as simulações.

As variáveis Average number of defects per page, Average number of defects detected per page, Process maturity level, Duration of inspection meeting (in minutes), Number of Inspectors per Team, Number of pages per inspection, Number of Parallel Teams, Size of documentation in pages, Number of Inspections e Inspection Technique, representam os parâmetros de entrada do modelo, ou seja, seus valores são determinados pelo usuário, de acordo com o cenário que ele deseja configurar.

Também fazem parte do modelo variáveis que "calculam" os efeitos dos parâmetros de entrada sobre a taxa de detecção de defeitos, são elas: Initial Quality of the Product, Efficiency of the inspectors, Impact factor of the duration on the performance, Impact factor of the number of inspectors on the performance, Impact

factor of the number of parallel teams on the performance, Inspection rate per pages e Efficiency of the inspectors in relation to the number of pages per inspection.

Já as variáveis que não influenciam nenhuma outra variável são utilizadas para análises dos resultados, que pode ser na forma de gráficos e tabelas, como é o caso da variável Defects found per person hour, esta representa o esforço gasto a partir do cenário configurado. Segundo (CLARCK, 2000) o esforço gasto com inspeções de software, incluindo as fases de preparação, reuniões e retrabalho geralmente consomem de 5 e 10% do esforço total do projeto. Pages not inspected e Defects not found são variáveis que também não influenciam nenhuma outra variável, servindo apenas para análise e conclusões.

No modelo há a possibilidade de simulação de 4 tipos diferentes de documentos de software: Requisitos de Software, Design, Design Detalhado e Código-fonte. Cada tipo de documento mencionado possui uma média diferente de defeitos por página (Average number of defects per Page) e também uma média de defeitos detectados (Average number of defects detected per page). Segundo (ROBERT, 1994) geralmente são mais utilizadas inspeções em documentos de códigos-fonte, seguido de inspeções em documento de Design, Requisitos e Casos de Testes. Experimentos revelam que a utilização de inspeções de requisitos, design, código fonte e testes, em média, diminuem 28,5%, 33,8%, 37,3% e 16,1% dos custos totais do software, respectivamente (ROBERT, 1994).

5 PAINEL DE CONTROLE PARA CONFIGURAÇÃO DE CENÁRIOS

O painel de controle é um recurso visual que tem como objetivo facilitar a interação dos gerentes de projetos com o modelo, oferecendo facilidades e usabilidade na configuração de cenários. Por meio dele é possível os usuários ajustarem os valores das variáveis de entrada do modelo de acordo com o cenário desejado, configurando características específicas da organização, da equipe e do tipo de inspeção que definem o contexto da simulação.

No painel de controle construído para o modelo são exibidos os respectivos valores máximos e mínimos que as variáveis de entrada podem atingir. Os valores das variáveis podem ser alterados por meio de controles deslizantes (slide bars) ou diretamente no combobox. O painel de controle oferece também um dispositivo de segurança caso o usuário entre com valores fora do permitido. Caso isso aconteça, o

valor da variável é definido com o valor mais próximo do valor desejado para simulação.

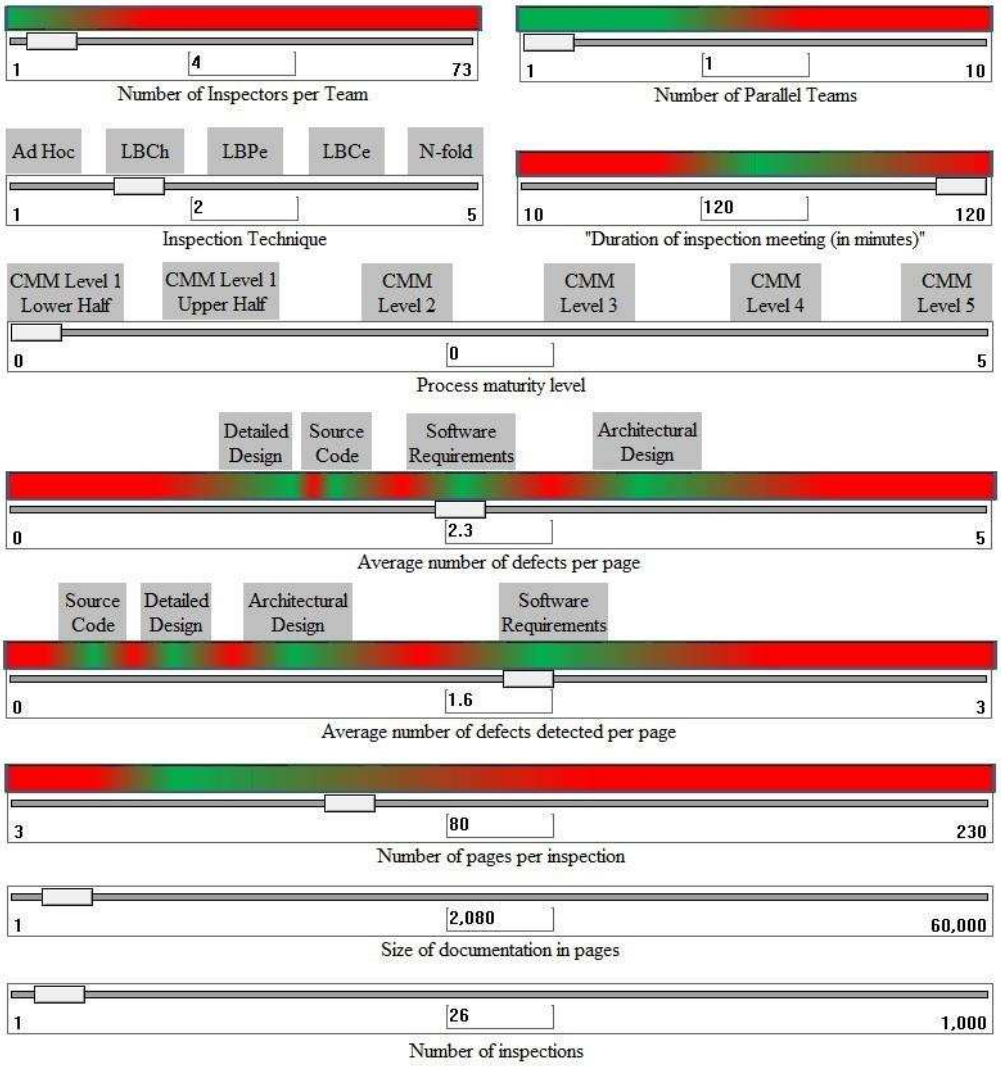

Figura 3. Painel de controle para configuração de cenários.

Como apresentado na Fig. 3, o painel de controle disponibiliza também de barras com as cores verdes e vermelhas para algumas variáveis. Verde indica valores dentro da faixa esperada e descrita na literatura, e vermelho indica valores fora desta faixa, que podem levar o sistema a desestabilizar. Os limites das variáveis foram definidos de acordo com as respectivas fontes utilizadas na definição do relacionamento de influência. Para as variáveis Average number of defects per page e Average number of defects detected per page as áreas verdes representam as médias de experimentos publicados na literatura em relação ao tipo de documento inspecionado e ao cenário base utilizado nas simulações (Fig. 3). Já as áreas verdes das variáveis Duration of inspection meeting (in minutes), Number of Inspectors per

Team, Number of pages per inspection, Number of Parallel Teams representam os valores recomendados em diversos experimentos da literatura, ou seja, ao sair dessa área verde possivelmente o gerente de projeto estará tomando decisões de forma ineficiente, desperdiçando recursos.

No painel de controle há a possibilidade de simulação de 5 técnicas de leitura, que segundo (LAITENBERGER, 2000) são as mais utilizadas pelas organizações: Ad Hoc, LBCh, LBPe, LBCe e N-fold.

A variável Process maturity level representa os níveis CMM de 1 a 5 que podem ser simulados. O modelo CMM avalia a maturidade das organizações no desenvolvimento de software e encontra-se como o modelo mais difundido em nível mundial.

6 SIMULAÇÕES, VALIDAÇÃO E USO DO MODELO NO APOIO À DECISÃO

Geralmente, simulações são usadas para determinar o efeito que uma alteração em uma parte do sistema pode causar no sistema como um todo. Por meio desse tipo de análise é possível avaliar o impacto que uma situação fora do comum pode causar no sistema. Esse tipo de estudo é chamado de análise de cenário (FORRESTER, 1980). Simulações também podem ser utilizadas para validação de hipóteses ou teorias. Pelo fato do ambiente utilizado ser virtual, o tempo e esforços gastos na execução das simulações são relativamente menores quando comparados a experimentos do mundo real.

De acordo com (BRATLEY, 1987), um método que pode ser realizado para validação de um modelo de simulação é o de alteração dos valores de entrada e analisar se esses valores fornecidos são condizentes com respostas no mundo real. (BRATLEY, 1987) recomenda que variem somente os dados de entrada que serão avaliados, os demais devem permanecer constantes. É comum nesse tipo de teste também ser realizado uma análise de sensibilidade, indicando ao modelador quais são os parâmetros de maior influência nos resultados.

6.1 Influência dos Inspetores no Processo

Diversos fatores podem influenciar na relação custo-eficiência (número de defeitos por unidade de tempo) de uma inspeção e dos tipos de defeitos encontrados.

Características relacionadas à experiência do inspetor ou à sua especialidade técnica no processo de desenvolvimento (analista, programador ou testador) podem afetar o resultado da inspeção. Os critérios para a escolha dos inspetores mais capacitados para a atividade de detecção de defeitos devem ser feitos considerando a relação entre suas habilidades e o contexto da inspeção (KALINOWSKI, 2004). Os resultados obtidos variando o número de inspetores podem ser observados analisando o gráfico da Fig. 4:

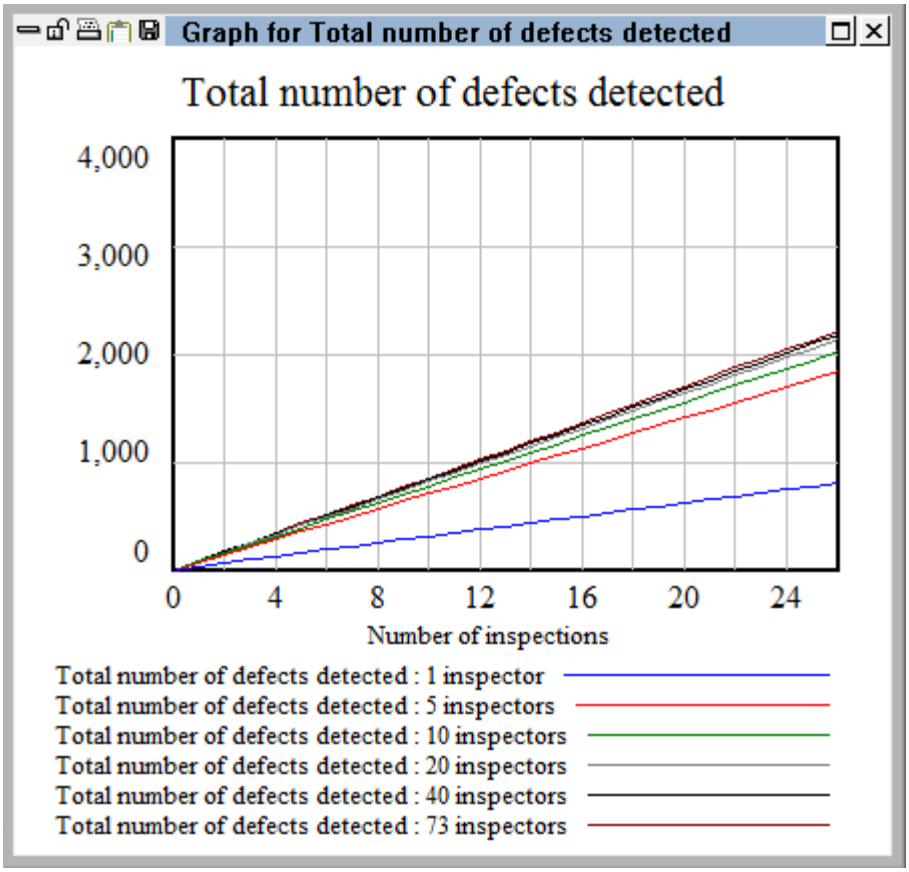

Figura 4. Total de defeitos detectados em relação ao número de inspetores.

Os resultados gerados com a simulação no modelo proposto neste artigo mostram que a variação do total de defeitos detectados é bastante significativa quando o número de inspetores é relativamente baixo.

Não há até hoje estudos que comprovem de fato o tamanho ideal da equipe de inspetores. Experimentos disponíveis na literatura variam nas suas recomendações. (MADACHY, 1996), por exemplo, recomenda entre 3 e 5 revisores. Para (KANTOROWITITZ, 1997), o tamanho da equipe de inspetores deve ser 3 ou 4. (PORTER, 1995) afirma que diminuindo o tamanho de equipes de inspetores de 4

para 2 pode reduzir significativamente os esforços sem que haja uma perda relevante na eficiência do processo.

6.2 Comparação da Eficiência das Técnicas de Leitura

Estudos apontam que a diferença dos resultados obtidos com a utilização das técnicas de leitura Ad hoc e Checklists, largamente aplicadas nas empresas, é relativamente pequena (DUNSMORE, 2000). Segundo (BASILI, 1996), inspeções de requisitos que se utilizem das técnicas de leitura baseadas em perspectiva encontrariam, em média, 35% mais defeitos que inspeções ad-hoc. Por outro lado, estes mesmos autores reconhecem que a aplicação da técnica requer um esforço 30% maior. Esses estudos deixa claro que cada técnica utilizada na inspeção de software possui níveis de eficiência e esforços bastante variados. Portanto, nem sempre a utilização da técnica com maiores índices de detecção de defeitos será a melhor escolha a se fazer. A eficiência na taxa de detecção de defeitos foi obtida dos estudos realizados por (KELLY, 1992), (KANTOROWITITZ, 1997), (BASILI, 1996) e (PORTER, 1995). Os resultados obtidos com as simulações realizadas no modelo proposto utilizando os diferentes tipos de técnicas de leitura podem ser vistos analisando o gráfico da Fig. 5 abaixo.

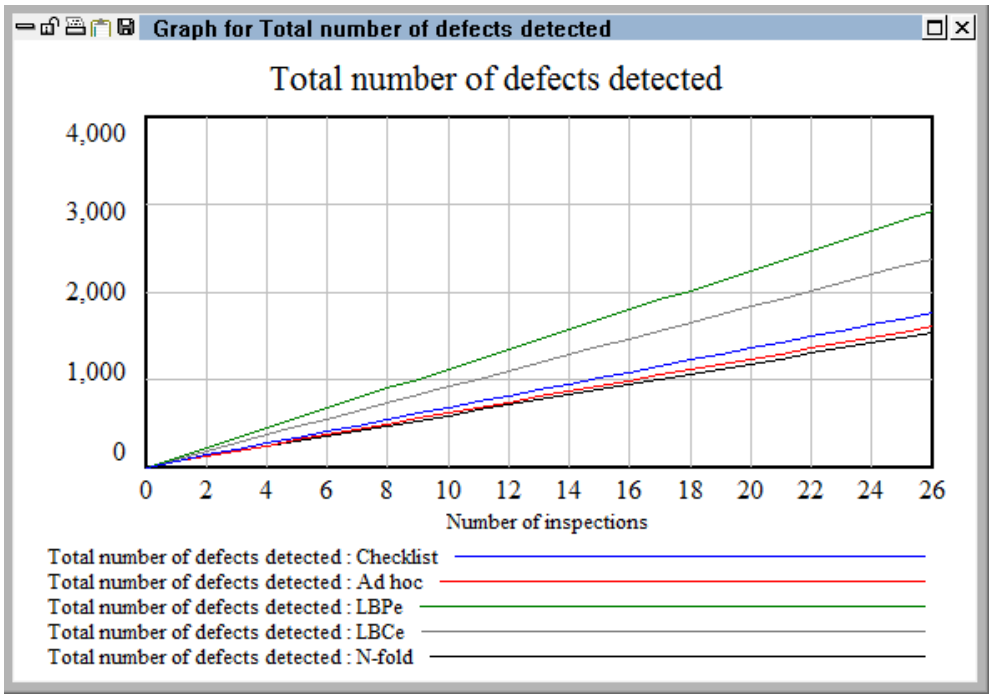

Figura 5. Número de defeitos encontrados em relação à técnica de leitura utilizada.

Detectando 2377 defeitos com a utilização da técnica de leitura baseada em cenários, observou-se 35% e 48% de eficiência a mais que as técnicas de leitura Checklist e Ad Hoc, respectivamente. Outra observação que pode ser notada ao analisar o gráfico é a superioridade da técnica LBPe em relação às demais, que segundo experimentos realizados no mundo real, é a técnica mais eficiente entre todas. Percebe-se que os resultados obtidos com as simulações das técnicas de leitura são condizentes com os resultados fornecidos na literatura.

6.3 Utilização do modelo no apoio à decisão

Um modelo de dinâmica de sistemas que aborde a atividade de inspeção em diferentes ciclos de vida do desenvolvimento de software permite que gerentes possam utilizá-lo como uma ferramenta de apoio para auxiliá-los na análise e tomada de decisão relativas aos projetos. A proposta deste trabalho busca contribuir para esse ambiente propício à ocorrência de defeitos, proporcionando aos gestores um tempo maior para tomada de decisão, bem como proporciona um melhor entendimento da tarefa de monitoramento de riscos como um todo. Através do modelo proposto neste artigo é possível simular o ambiente, estabelecendo estimativas da eficiência do processo de inspeção.

O modelo proposto pode ser utilizado para coordenar e apoiar a inspeção de 4 tipos de artefato de software (documentos de requisitos, design, design detalhado e código-fonte). Através do modelo, gerentes de projetos, podem configurar e simular cenários para auxiliá-los na análise e tomadas de decisão relativas às características da empresa, permitindo testar situações novas, antes de colocá-las em prática.

O principal benefício esperado pela simulação no modelo proposto é fornecer para o gerente de projeto e para a equipe de desenvolvimento um diagnóstico “a priori” sobre os possíveis impactos da utilização de inspeções nos diferentes ciclos de vida do desenvolvimento do projeto.

Outra motivação para a utilização do modelo é a possibilidade de realizar a análise do impacto de modificações no modelo de processo de inspeção já utilizado na organização, tendo em vista que estas modificações possuem, em geral, um grande risco e alto custo. Com a simulação é possível avaliar e reduzir os riscos destas mudanças antes de implantá-las em um projeto real. Os gerentes de projetos

podem reduzir o tempo, custo e o risco dos experimentos devido ao ambiente ser virtual e assim não possuir riscos reais.

A decisão de qual cenário aplicar depende de fatores que influenciam no projeto e na empresa em questão, principalmente da disponibilidade de profissionais, prazo do projeto e recursos disponíveis. O modelo em si não mostra qual a decisão gerencial que deve ser tomada, mas auxilia os gerentes a enxergarem o contexto dos cenários.

A partir do modelo apresentado, é possível verificar, por exemplo, como os 5 níveis de maturidade do CMM afeta a taxa de detecção de defeitos, a eficiência dos inspetores e a qualidade dos documentos que são inspecionados. Utilizando o modelo proposto, é possível simular também a utilização de diferentes tipos de técnicas de leitura. Com isso, os gerentes de projeto podem avaliar a eficiência e o esforço exigido por cada uma das técnicas antes de colocá-las em prática.

7 CONSIDERAÇÕES FINAIS

O modelo de dinâmica de sistemas foi construído com base em dados publicados na literatura. Isso torna a estrutura do modelo fundamentada no conhecimento disponibilizado pela comunidade científica, garantindo uma maior confiabilidade dos resultados obtidos a partir das simulações realizadas. Os resultados das simulações se mostraram compatíveis com as principais referências da literatura sobre inspeções de software, o que representou mais uma validação para o trabalho.

O modelo de dinâmica de sistemas apresentado permite verificar, por meio de simulações, os impactos das variáveis que influenciam fortemente na qualidade da inspeção em documentos de projetos de software. Esse modelo pode ser utilizado como uma ferramenta de apoio à tomada de decisão, economizando recursos e permitindo reproduzir cenários nos quais seria caro ou perigoso experimentar na realidade, favorecendo a adoção de medidas preventivas.

A partir do modelo é possível estimar o total de defeitos distintos do produto e ao mesmo tempo acesso a uma estimativa da efetividade do sistema de inspeção, medida que é utilizada por algumas empresas como um critério para continuidade ou não da inspeção, ou para reinspeção de unidades de produtos.

2.4 Otimizando recursos no processo de inspeção de software: uma abordagem utilizando simulação com dinâmica de sistemas

Jailton S. Coelho1, José Luis Braga1, Bernardo Giori Ambrósio2

ABSTRACT

Reparar um defeito de software pode custar até 100 vezes mais caro, caso ele não seja encontrado o mais próximo possível de onde foi cometido. A inspeção de software é uma técnica que pode ser usada para ajudar a detectar defeitos nas fases iniciais do processo de desenvolvimento, evitando que esses defeitos sejam propagados para as fases seguintes. O custo/benefício de inspeções pode se tornar bastante significativo, caso as inspeções sejam realizadas de forma eficiente. Por ser influenciada por muitos fatores de qualidade, a análise do contexto da inspeção como um todo pode se tornar complexa. Gerentes de projeto deixam de utilizar a inspeção por falta de dados e estudos sobre os reais benefícios que ela pode gerar. Este artigo apresenta um modelo de dinâmica de sistemas, para apoiar decisões sobre esse ponto de vista. As variáveis que fazem parte do modelo são quantificadas com base em experimentos reais ou empíricos disponibilizados na literatura, tornando os resultados do modelo próximos do que seria obtido no mundo real. O modelo permite reproduzir cenários nos quais seria caro ou perigoso experimentar no mundo real, sendo possível antever os impactos que a inspeção pode trazer no processo de desenvolvimento.

Keywords: inspeção de software, dinâmica de sistemas, detecção de defeitos.

1 INTRODUÇÃO

A qualidade de um produto de software não pode ser introduzida após a finalização do mesmo, deve ser acompanhado e garantido durante todo o processo de desenvolvimento (RUSSEL, 1991). A participação humana é um fator presente nos projetos de software, o que justifica a ocorrência de defeitos nos sistemas de software mesmo quando as melhores técnicas e procedimentos são empregados no seu desenvolvimento (KNIGHT, 1993).

Em torno de 40 a 50% do esforço no desenvolvimento de software é gasto em retrabalho e grande parte desse esforço é gasto com a remoção de defeitos, exigindo, em algumas situações, mudanças significativas na arquitetura do projeto (SHULL, 1991). O custo para remover um defeito de software aumenta na medida em que o processo de desenvolvimento progride para os estágios finais (SHULL, 1991). Encontrar e corrigir um defeito de software pode ser até 100 vezes mais caro, caso ele não seja encontrado próximo da fase em que foi cometido no processo de desenvolvimento (SHULL, 1991). (LAITENBERGER, 1991) considera defeito como qualquer desvio nas propriedades de qualidade do software, o que inclui o não atendimento aos requisitos estabelecidos.

Atividades de prevenção e detecção de defeitos devem ser introduzidas ao longo de todo o processo de desenvolvimento com o objetivo de diminuir os custos de correção nas fases finais. Alocar recursos para a realização de revisões de artefatos de software, tão logo estejam finalizados, pode contribuir para reduzir o retrabalho e melhorar a qualidade dos produtos (HERMSDORF, 2011). Encontrar um defeito o mais cedo possível pode gerar uma enorme contribuição para o projeto, do ponto de vista econômico e de qualidade.

A inspeção de software é um tipo específico de revisão que pode ser aplicado em artefatos de software e possui um processo de detecção de defeitos rigoroso e bem definido (PORTER, 1995). Segundo (LAITENBERGER, 1991), inspeções têm a função de detectar defeitos antes que a fase de teste seja iniciada, contribuindo para a melhoria da qualidade geral do software em relação ao seu orçamento e benefícios de tempo. Apesar desses benefícios, em geral os gerentes de projetos não adotam a inspeção no processo de desenvolvimento de software devido ao receio de não obterem um retorno satisfatório (HERMSDORF, 2011), que decorre da falta de ferramentas para auxiliar na obtenção de uma estimativa do retorno que será obtido com a alocação de recursos para realizar tais atividades.

Esse artigo apresenta um modelo de dinâmica de sistemas (FORRESTER, 1980; MADACHY, 1996; STERMAN, 2000) que permite avaliar os impactos da alocação de recursos para a realização de inspeções em várias fases do processo de desenvolvimento de software. O modelo permite simular e analisar os resultados da política gerencial quanto à realização de inspeções antes de aplicá-la em um projeto real. O modelo pode ser usado como uma ferramenta de apoio à decisão permitindo simular diversos cenários de acordo com as características da empresa, equipe e

projeto, sem que seja necessário realizar experimentos em projetos reais, o que seria caro e demandaria tempo.

No modelo foram incluídas variáveis que influenciam diretamente na qualidade do processo de inspeção, bem como no retorno obtido ao alocar recursos para a sua realização, dentre as quais tem-se: tamanho da equipe de inspetores, técnica de leitura, nível CMMI da empresa, tipo e tamanho do documento, entre outras. Essas variáveis e os relacionamentos de influência existentes entre as mesmas foram identificados e calibrados com base em experimentos reais ou empíricos publicados na literatura, o que garante a consistência do modelo e o torna coerente com os conhecimentos difundidos pela Engenharia de Software.

Este artigo está estruturado como se segue: a Seção 2 aborda os conceitos envolvidos no contexto desse trabalho e descreve alguns trabalhos correlatos, a Seção 3 descreve o processo de construção do modelo, a Seção 4 apresenta as simulações realizadas com o modelo construído para avaliar os níveis de influência dos fatores que causam impacto na atividade de inspeção e a Seção 5 descreve as conclusões e os trabalhos futuros.

2 CONTEXTO

Este trabalho classifica-se como pesquisa aplicada ou tecnológica, pois utiliza conhecimentos básicos, disponíveis na literatura, para gerar novos conhecimentos para aplicação prática. Esse tipo de pesquisa se baseia em um estudo observacional e, no âmbito dos procedimentos de pesquisa, o mesmo é caracterizado pela extensa pesquisa bibliográfica (JUNG, 2004).

2.1 Inspeção de Software

A inspeção é um tipo particular de revisão que contribui para garantir a qualidade do produto de software. Todas as etapas do processo de desenvolvimento de software são suscetíveis à incorporação de defeitos, que podem ser detectados pela inspeção e posteriormente removidos.

O processo básico da inspeção consiste de seis etapas consecutivas (KELLY, 1992):

- Planejamento. Desempenhado por um moderador cuja função é definir o contexto da inspeção (descrição da inspeção, técnica de leitura a ser utilizada

na atividade de detecção de defeitos, documento a ser inspecionado, autor do documento, dentre outros), selecionar os inspetores e distribuir o material a ser inspecionado.

- Detecção de Defeitos. Os inspetores selecionados têm a tarefa de encontrar defeitos no documento entregue a eles e, ao longo da inspeção, produzir uma lista dos possíveis defeitos.
- Coleção de Defeitos. O moderador tem a função de agrupar as listas de defeitos dos inspetores, eliminando os defeitos repetidos, ou seja, que foram encontrados por mais de um inspetor, mantendo só um registro para cada defeito.
- Discriminação de Defeitos. O moderador juntamente com o autor do documento e os inspetores discutem os defeitos a fim de classificá-los como defeito de fato ou falso positivo. Os falsos positivos são eliminados da lista de defeitos, permanecendo apenas os defeitos reais.
- Retrabalho. Os defeitos que fazem parte da lista são corrigidos pelo autor do documento e ao término é produzido um relatório de correção dos defeitos.
- Continuação. É decidido pelo moderador se uma nova inspeção deve ou não ser realizada.

A inspeção utiliza-se da revisão baseada na leitura e entendimento de um artefato de software a fim de encontrar defeitos. Muitos desenvolvedores possuem habilidades para escrever documentos de requisitos e códigos de software, mas não é oferecido a eles nenhum suporte relacionado à leitura e revisão adequada desses artefatos. Uma das soluções é então fazer uso de técnicas de leitura.

Técnicas de leitura contribuem para o aumento da compreensão sobre algum artefato de software. Esta compreensão deve ser suficiente a ponto de permitir que os inspetores identifiquem tanto as informações importantes para a execução de uma determinada tarefa, como a relação destas informações com o problema que está sendo tratado (HERMSDORF, 2011). Algumas das principais técnicas de leitura são (LAITENBERGER, 1991):

- Ad-hoc: não oferece nenhum suporte técnico de como detectar defeitos em um artefato de software. Neste caso, a detecção de defeitos depende totalmente da habilidade, do conhecimento e da experiência de um inspetor.

- Checklist (LBCh): baseia-se em uma série de perguntas (frequentemente questões cuja resposta é sim ou não) sobre assuntos do documento a ser inspecionado.
- Cenário (LBCe): faz uso da noção de cenários que fornecem orientação personalizada para inspetores sobre como detectar defeitos. Um cenário pode ser um conjunto de perguntas ou uma descrição mais detalhada para um inspetor sobre a forma de realizar a análise do documento.
- Stepwise Abstraction (SA): fornece instruções de leitura mais estruturadas e precisas para os artefatos de código. Ao ler uma sequência de declarações no código, o inspetor abstrai uma função de calcular dessas declarações. Este procedimento é repetido até que a última função do artefato de código inspecionado seja captada, comparando no final essas funções com a especificação.
- Análise de Pontos de Função (APF): define um sistema em termos de suas entradas, arquivos, consultas e saída. Os cenários de pontos de função são desenvolvidos em torno desses itens. Um cenário de pontos de função consiste em perguntas e direciona o foco de um inspetor para um item de ponto de função específico dentro do documento de requisitos inspecionado.
- Perspectiva (LBPe): esta técnica baseia-se na ideia de que um produto de software deve ser inspecionado a partir da perspectiva dos diferentes stakeholders. As perspectivas desta técnica são elaboradas de modo que os inspetores são forçados a enxergar o ponto de vista do usuário, do projetista e do testador do sistema, utilizando a perspectiva correspondente.
- N-fold: o método de leitura N-fold consiste na replicação do processo de realização de inspeções formais para detecção de defeitos usando N equipes trabalhando em paralelo com um único moderador que é o responsável por coordenar e reunir os resultados. Essa técnica é aplicada considerando a hipótese de que a existência de equipes diferentes melhora a eficiência do processo, detectando defeitos que não seriam encontrados por uma única equipe de inspetores.

2.2 Dinâmica de Sistemas

A dinâmica de sistemas é uma técnica descritiva utilizada na modelagem e simulação de sistemas, que permite levar em consideração as variações dinâmicas dos problemas, sendo fortemente baseada no pensamento e análise sistêmicos e na teoria matemática dos Sistemas Dinâmicos. A Dinâmica de Sistemas permite a compreensão de como as políticas adotadas, ou a própria estrutura do sistema, afetam ou determinam o seu comportamento dinâmico, permitindo prever comportamentos futuros e possíveis colapsos. O uso dessa técnica permite a análise e compreensão de problemas e situações de maneira integrada e interconectada, deixando claras as relações existentes entre as variáveis de decisão (AHMED, 2009).

Um modelo de dinâmica de sistemas possui dois elementos principais: os estoques, que são os recursos acumuláveis do sistema, e os fluxos ou taxas, que são funções que representam as decisões ou políticas da empresa com relação aos usos e acúmulos dos estoques ou recursos. Há também os conversores, ou variáveis simples, que são os elementos do modelo que exercem influência sobre os valores dos fluxos (MADACHY, 1996). Esses elementos são ilustrados na Fig. 1.

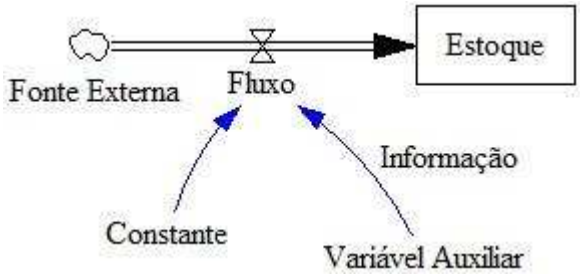

Fig. 1. Elementos básicos de um modelo genérico de estoque e fluxo (MADACHY, 1996).

Segue a descrição dos elementos da Fig. 1:

- Estoques: representam os acúmulos ou decrementos ao longo do tempo. Seus valores são influenciados pelos fluxos de entrada e/ou de saída.
- Fonte externa: são recursos ou repositórios infinitos utilizados pelos fluxos que não são especificados no modelo e não fazem parte do contexto da simulação.
- Fluxos: podem ser considerados as "ações" no sistema. Eles influenciam diretamente os estoques e podem representar as decisões e políticas adotadas.
- Conversores (ou variáveis auxiliares): são utilizados na elaboração dos detalhes da estrutura do modelo e para representar as variáveis que

influenciam os fluxos. Eles se relacionam com os fluxos através de setas unidirecionais.

- Informação (setas unidirecionais): são utilizadas para representar fluxos de informação e interligar as variáveis do modelo. Estas setas permitem construir loops de realimentação (feedback) no modelo (STERMAN, 2000).
- Constantes: representam as variáveis que não se alteram durante toda simulação.

2.3 Trabalhos correlatos

Um dos modelos de dinâmica de sistemas mais difundidos na área de Engenharia de Software é o modelo proposto por (HAMID, 1990) para simular o desenvolvimento de projetos de software. Esse modelo considera os seguintes aspectos de um projeto: Gerência de Recursos Humanos, Produção de Software, Desenvolvimento de Software, Garantia de Qualidade e Retrabalho, Testes e Controle.

(MADACHY, 1996) usou a disciplina dinâmica de sistemas com o objetivo de modelar os efeitos da realização de inspeções formais em diversas fases do ciclo de desenvolvimento de softwares. Foram comparados questões como impactos de custo, prazo e qualidade para diferentes taxas de inspeção. O modelo trata também da correlação entre fluxos e tarefas, erros e recursos humanos entre as diversas fases do desenvolvimento de projetos de software.

O modelo proposto neste artigo também utiliza a dinâmica de sistemas. Entretanto, o modelo proposto simplifica a visão dos fatores que mais influenciam na qualidade e eficiência de inspeções, permite aos usuários interagir com o modelo e é focado na etapa de detecção de defeitos, que é uma das seis etapas da atividade de inspeção.

3 CONSTRUÇÃO DO MODELO

O modelo de dinâmica de sistemas apresentado neste trabalho foi construído seguindo o método proposto por (AMBROSIO, 2011), constituído por três etapas subdivididas em um ou mais passos. Na primeira etapa são identificadas as variáveis envolvidas no processo ou problema que se pretende modelar. Em seguida, os relacionamentos existentes entre essas variáveis devem ser identificados, o que pode

ser feito com o auxílio dos diagramas de influência, que permitem construir modelos simples e de fácil entendimento e são muito utilizados em análise sistêmica. Os diagramas de influência permitem modelar relações de influência entre as variáveis, facilitando a identificação dos laços de realimentação (STERMAN, 2000). Na segunda etapa os diagramas de influência são mapeados em modelos de dinâmica de sistemas seguindo algumas regras estabelecidas no método proposto. Na terceira etapa, os modelos de dinâmica de sistemas são refinados e os relacionamentos entre seus elementos são quantificados e testados, até que seja obtido um modelo finalizado e validado.

3.1 Seleção das Variáveis

Inicialmente foram identificadas as variáveis que representam os fatores que mais influenciam a atividade de inspeção de artefatos de projetos de software, que segundo (LAITENBERGER, 1991) são: as características do time, o esforço, a técnica de leitura, a organização da atividade de detecção de defeitos e as características do produto. Essas variáveis são ilustradas na Fig. 2.

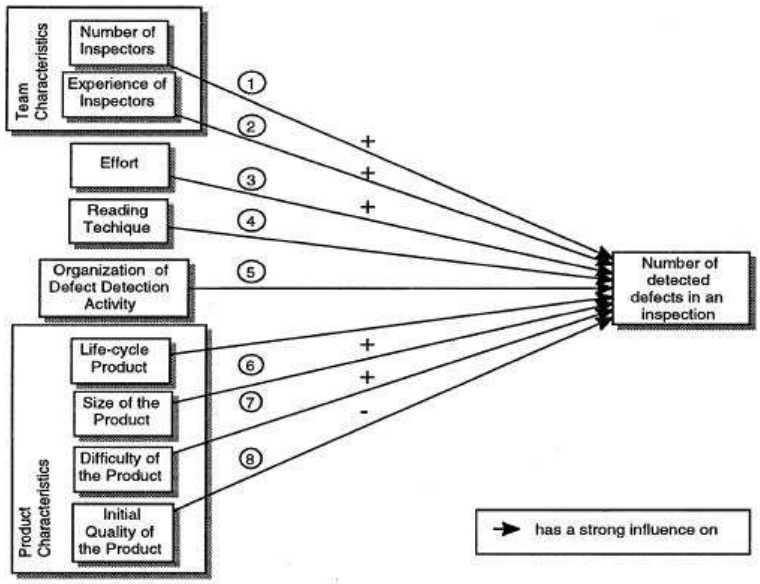

Fig. 2. Principais fatores de forte influência no desempenho do processo de inspeção (LAITENBERGER, 1991).

As características do time referem-se à quantidade e à experiência dos inspetores envolvidos no processo. O esforço representa o custo total necessário para realização das inspeções. A técnica de leitura, que pode ser diferente em cada fase do ciclo de vida do produto, refere-se ao modo como será realizada a leitura do artefato

para detectar os defeitos. A organização da atividade de detecção de defeitos refere-se ao número de inspeções a serem realizadas, à técnica de leitura e ao número de inspetores que farão parte da equipe. As características do produto são representadas pelo ciclo de vida, tamanho, complexidade e qualidade inicial do produto. O ciclo de vida se refere à fase de desenvolvimento do projeto. O tamanho do documento se refere ao número de páginas que ele possui. A dificuldade do produto se refere à complexidade de entendimento do mesmo, ou seja, quanto maior a complexidade maior será a dificuldade de detectar um defeito. A qualidade inicial do produto se refere ao número de defeitos que podem estar contidos no artefato de software, ou seja, quanto maior for a qualidade menor será o número de defeitos existentes.

3.2 Modelo de Dinâmica de Sistemas

Após identificar os relacionamentos entre as variáveis, com base em dados extraídos da literatura, foram construídos diagramas de influência para representar as relações de causa e efeito entre as variáveis. Em seguida, esses diagramas de influência foram mapeados em um modelo de dinâmica de sistemas, utilizando-se as seguintes regras:

- Variáveis que representam algo que sofre acúmulos ou decrementos ao longo do tempo foram mapeadas em estoques;
- Variáveis que influenciam diretamente outra variável previamente mapeada em um estoque foram mapeadas em fluxos;
- Variáveis que representam os parâmetros de entrada para realizar a simulação do modelo foram convertidas em conversores constantes, ou seja, seu valor não é alterado durante toda simulação;
- Variáveis não classificadas em nenhuma categoria anterior (estoques, fluxos e constantes) foram mapeadas em conversores.

Para construir o modelo de dinâmica de sistemas foi utilizada a ferramenta Vensim (VENSIM, 2010), em sua versão gratuita para uso acadêmico. O modelo obtido é apresentado na Fig. 3.

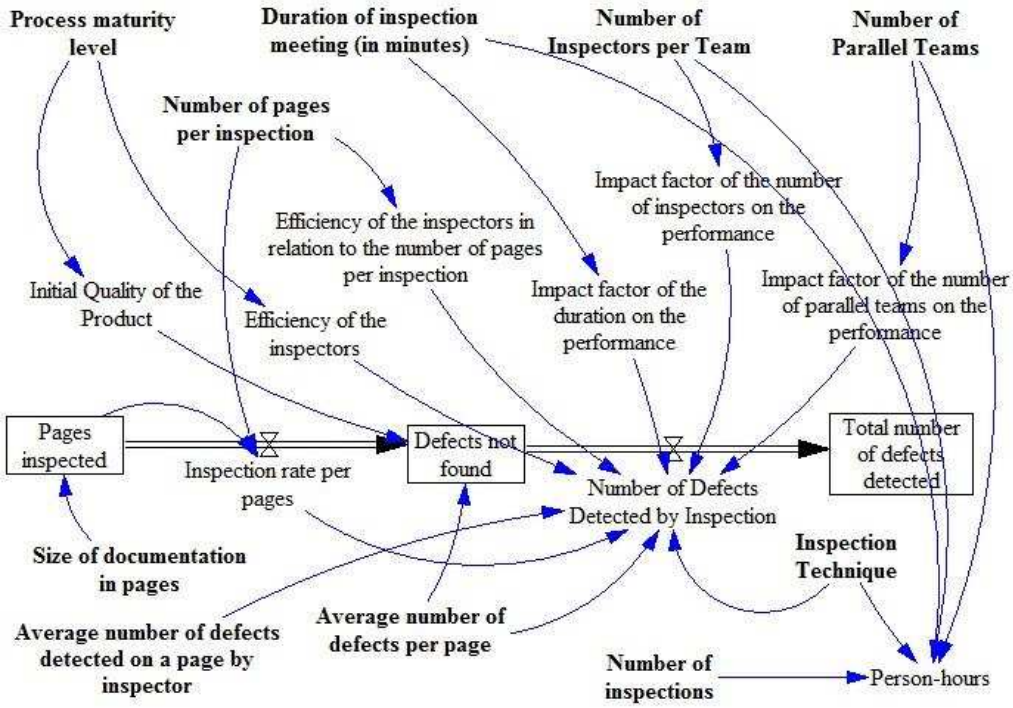

Fig. 3. Modelo estoque e fluxo da Dinâmica de Sistemas.

A variável Process maturity level representa o nível CMMI da empresa. Ela influencia positivamente as variáveis Efficiency of the inspectors e Initial Quality of the Product, que representam respectivamente a produtividade dos inspetores e a qualidade inicial do artefato que será inspecionado. O relacionamento de influência da variável Process maturity level sobre as variáveis Efficiency of the inspectors e Initial Quality of the Product foi quantificado com base nos estudos realizados por (GIBSON, 2006). Variando o nível CMMI da empresa do mais baixo para o mais alto, a produtividade dos inspetores e a qualidade final do produto podem aumentar em até 329% e 132%, respectivamente.

A variável Number of pages per inspection representa o número de páginas do documento alocadas por inspeção e influencia positivamente na taxa de inspeção por páginas, representada pelo fluxo Inspection rate per pages, e negativamente na eficiência dos inspetores, representada pela variável Efficiency of the inspectors in relation to the number of pages per inspection. O relacionamento de influência da variável Number of pages per inspection sobre a eficiência dos inspetores foi modelada como uma variável-gráfica baseada nos experimentos realizados por (KELLY, 1992). Uma variável-gráfica é utilizado quando não se conhece a equação matemática que define o relacionamento entre duas variáveis e permite quantificar o

relacionamento entre variáveis por meio de um esboço do gráfico correspondente à função matemática que quantifica o relacionamento.

O tempo de reunião da inspeção representado pela variável Duration of inspection meeting influencia não só na taxa de defeitos detectados mas também nos custos. A variável Impact factor of the duration on the performance é uma variável-gráfica que representa o esboço do gráfico correspondente à função matemática que quantifica o relacionamento de influência do tempo de inspeção sobre a taxa de defeitos detectados. O esboço do gráfico foi feito com base nos estudos realizados por (DUNSMORE, 2000) que recomenda que o tempo de reunião da inspeção não ultrapasse 2 horas.

A variável Number of Inspectors per Team representa o tamanho da equipe de inspetores responsável pela atividade de detecção de defeitos. O relacionamento de influência sobre a taxa de defeitos detectados é definido pela variável-gráfica Impact factor of the number of inspectors on the performance. O esboço do gráfico foi feito com base nos experimentos empíricos realizados por (WALIA, 2008), que chegou a simular o processo de inspeção com 73 inspetores.

A variável Number of Parallel Teams representa o número de equipes de inspetores paralelas que serão utilizadas pela técnica de leitura N-fold. O esboço do gráfico que quantifica o relacionamento de influência do número de equipes paralelas sobre a taxa de detecção de defeitos é representado pela variável-gráfica Impact factor of the number of parallel teams on the performance, e foi modelado baseando-se nos experimentos realizados por (KANTOROWITITZ, 1997), que simulou o processo de inspeção com até 10 equipes paralelas de inspetores, aumentando o número de defeitos detectados em até 100%.

O tamanho do documento que será simulado é representado pela variável Size of documentation in pages que pode ter no máximo 60 mil páginas. Esse limite é baseado nos estudos de (GIBSON, 2006) que cita documentos de código-fonte com o mesmo tamanho adotado no modelo.

A variável Number of inspections representa o número de inspeções que serão realizadas no documento. Já a técnica de leitura utilizada nas simulações é determinada pela variável Inspection Technique. A variável Person-hours representa o esforço total necessário para que as inspeções sejam realizadas. As variáveis Average number of defects per Page e Average number of defects detected on a page

by inspector representam respectivamente a média de defeitos por página e a média de defeitos detectados em uma página por um único inspetor.

O fluxo Rate of the number of defects detected by inspection representa o número de defeitos que são detectados por inspeção. O fluxo Inspection rate per pages representa o número de páginas alocadas para cada inspeção. O estoque Total number of defects detected representa a soma total dos defeitos detectados pelas inspeções e o estoque Defects not found representa o total de defeitos que não foram detectados com as inspeções. O estoque Pages inspected representa o total de páginas do documento que será inspecionado.

3.3 Painel de Controle

O painel de controle é uma ferramenta que permite aos usuários interagir com o modelo e realizar os ajustes para configurar os cenários a serem simulados. Para o modelo descrito neste artigo, foi definido um painel de controle, apresentado na Fig. 4, que permite aos gerentes de projetos ajustar os valores das variáveis de entrada do modelo de acordo com as características específicas da organização, da equipe e do tipo de inspeção que definem o contexto do cenário que se deseja simular.

Cada variável de entrada do modelo pode assumir valores dentro de uma faixa contínua que possui um valor mínimo e um valor máximo. Os valores das variáveis podem ser alterados por meio de controles deslizantes (slide bars) ou manualmente. Caso o usuário informe um valor inválido para alguma variável, esta receberá automaticamente o valor mais próximo do valor informado. Ao utilizar o painel de controle, o usuário deve ter o cuidado de não informar uma combinação de valores incompatíveis para as variáveis, para que não ocorra desestabilização do modelo e não sejam gerados resultados incoerentes com a realidade. O tomador de decisão deve conhecer muito bem o problema e as combinações aceitáveis de valores para as variáveis.

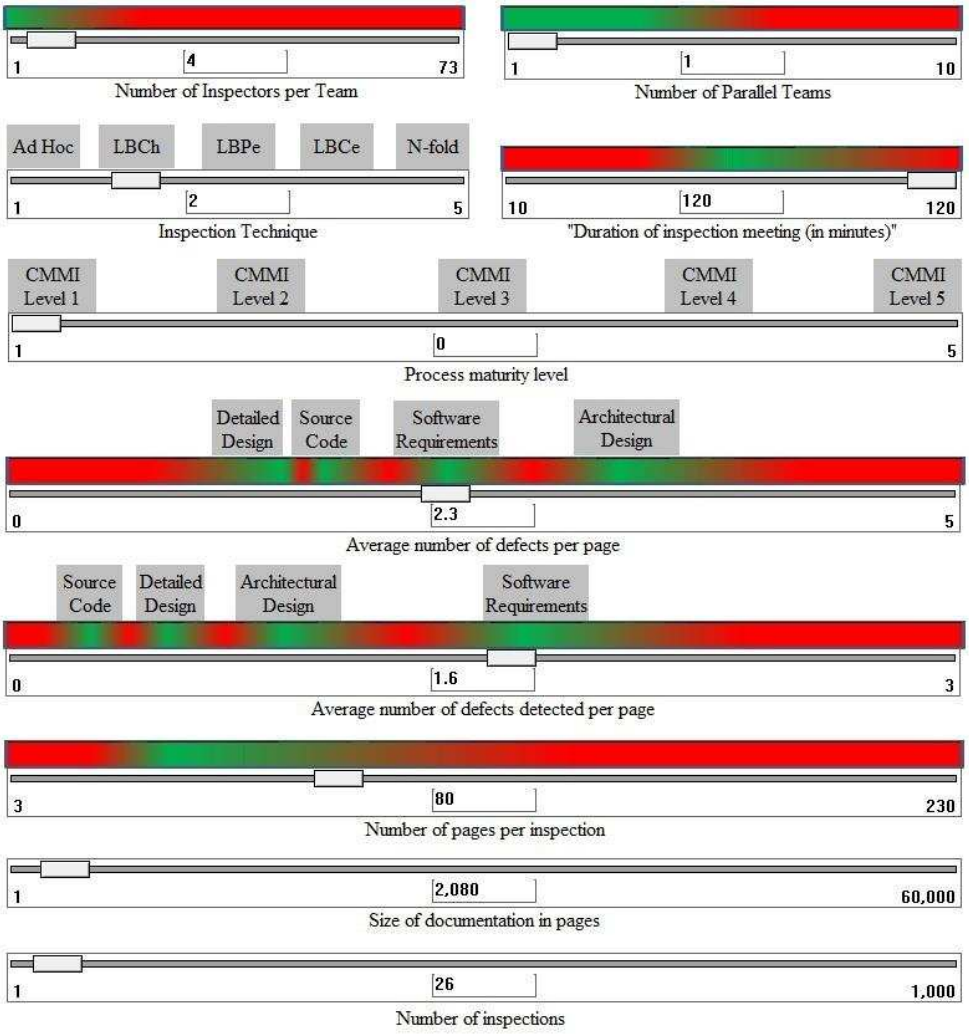

Fig. 4. Painel de controle para configurar cenários.

Algumas variáveis de entrada no painel de controle do modelo têm a sua faixa de valores permitidos representada por uma barra com a cor variando de verde a vermelho. Os valores próximos à cor verde estão dentro da faixa esperada e descrita na literatura; enquanto que os valores próximos à cor vermelha podem levar à desestabilização do modelo. Os limites das variáveis foram definidos de acordo com as respectivas fontes utilizadas na definição do relacionamento de influência. Para as variáveis Average number of defects per page e Average number of defects detected on a page by inspector os valores próximos à cor verde representam os valores médios obtidos em experimentos publicados na literatura considerando cada tipo de documento inspecionado. Já para as variáveis Duration of inspection meeting (in minutes), Number of Inspectors per Team, Number of pages per inspection e Number of Parallel Teams, os valores próximos à cor verde são os recomendados em diversos experimentos publicados na literatura, ou seja, ao definir valores fora dessa

faixa os gerentes de projetos estarão simulando cenários que representam situações reais onde recursos seriam desperdiçados na realização das inspeções. A variável Inspection Technique representa as 5 técnicas de leitura que podem ser simuladas: Ad Hoc, LBCh, LBPe, LBCe e N-fold. E por último, a variável Process maturity level representa os 5 níveis CMMI (GIBSON, 2006) que podem ser determinados nas simulações.

4 SIMULAÇÕES E ANÁLISES DOS RESULTADOS

As seções seguintes descrevem os resultados de simulações realizadas com o modelo de dinâmica de sistemas construído. Inicialmente foi definido e simulado um cenário base e posteriormente foram realizadas simulações de outros cenários obtidos a partir de alterações de alguns valores definidos para o cenário base. Os resultados obtidos permitiram avaliar antecipadamente os impactos que as variáveis de entrada do modelo exercem sobre o número de defeitos detectados e sobre os custos das inspeções. Os valores assumidos pelas variáveis do cenário base apresentados na Tab. 1 representam valores médios para um cenário real, e foram baseados nos experimentos de (KELLY, 1992) e (GIBSON, 2006). Este cenário será utilizado nas simulações apresentadas a seguir, com variações em algumas das variáveis cuja influência se deseja visualizar.

Tabela 1. Valores das variáveis utilizadas no cenário base.

Variável	Valor
Number of pages	2000
Average number of defects per Page	1.6
Average number of defects detected by inspector	0.2
Number of pages per inspection	40
Number of inspections	50
Process maturity level	CMMI Level 1
Inspection Technique	Checklist
Duration of inspection meeting	120 minutes
Number of parallel teams	1
Number of inspectors per team	4

4.1 Influência do Número de Equipes de Inspetores Paralelas

Adicionar equipes de inspetores ao processo de inspeção pode aumentar de forma significativa a taxa de detecção de defeitos, considerando que novas equipes

encontrariam defeitos que não seriam detectados pelas equipes já existentes (KANTOROWITITZ, 1997). O gráfico da Fig. 5 mostra os resultados das simulações ao considerar diferentes quantidades de equipes paralelas e permite analisar o impacto da adição de times de inspetores sobre o número de defeitos detectados e sobre os custos.

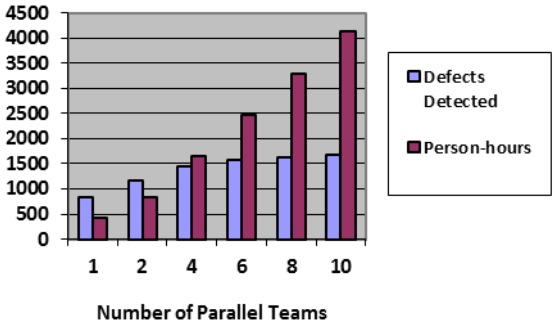

Fig. 5. Total de defeitos detectados e custo gasto em relação ao número de equipes paralelas.

O gráfico da Fig. 5 mostra que ao utilizar 1 equipe de inspetores foram detectados 833 defeitos, sendo gastos 412 homens-horas. Entretanto, ao utilizar 10 equipes paralelas foram detectados 1674 defeitos, necessitando de 4120 homens-horas. Observa-se que o aumento no número de equipes paralelas de inspetores não implica em um aumento proporcional do número de defeitos detectados. Portanto, a escolha do melhor valor para a quantidade de equipes depende da disponibilidade de recursos e do conhecimento do problema.

4.2 Influência do Tamanho da Equipe de Inspetores

Similarmente à variável Number of Parallel Teams, o tamanho da equipe de inspetores também influencia de forma significativa o número de defeitos encontrados e os custos. Entretanto, ao adicionar um inspetor à equipe, além de aumentar a taxa de detecção de defeitos, o tempo total gasto para inspecionar o documento diminui, já que uma parte do documento é atribuída a esse novo inspetor. Os resultados obtidos ao variar o número de inspetores podem ser observados no gráfico da Fig. 6.

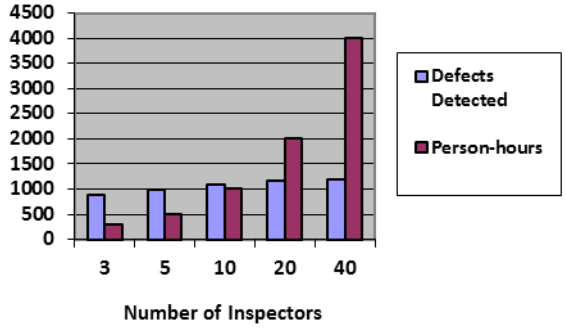

Fig. 6. Total de defeitos detectados e custo gasto em relação ao número de inspetores.

O gráfico da Fig. 6 mostra que ao utilizar 3 inspetores foram detectados 896 defeitos e foram gastos 300 homens-horas. E ao utilizar 10 inspetores, o número de defeitos detectados foi 1098 e o custo foi de 1000 homens-horas. É possível observar também que o crescimento do número de defeitos detectados ao utilizar mais de 20 inspetores é relativamente baixo, enquanto que os custos aumentam significativamente. Logo, a escolha do número de inspetores a serem utilizados na atividade de detecção de defeitos depende do total de recursos que podem ser gastos, observado o limite a partir do qual não há melhora nos resultados.

Experimentos disponíveis na literatura variam nas suas recomendações quanto ao tamanho ideal da equipe de inspetores. (MADACHY, 1996) recomenda entre 3 e 5 revisores, e para (KANTOROWITITZ, 1997), o ideal deve ser 3 ou 4.

4.3 Influência da Técnica de Leitura

Segundo (BASILI, 1996), inspeções realizadas em artefatos da fase de requisitos com o uso das técnicas de leitura baseadas em perspectiva encontrariam, em média, 35% mais defeitos que inspeções ad-hoc. Por outro lado, este mesmo autor reconhece que a aplicação da técnica requer um esforço 30% maior. Esse estudo deixa claro que cada técnica utilizada na inspeção de software possui níveis de eficiência e esforços bastante variados. Portanto, nem sempre a utilização da técnica com maiores índices de detecção de defeitos será a melhor escolha para o projeto. Os resultados obtidos ao simular o uso dos diferentes tipos de técnicas de leitura podem ser vistos analisando o gráfico da Fig. 7.

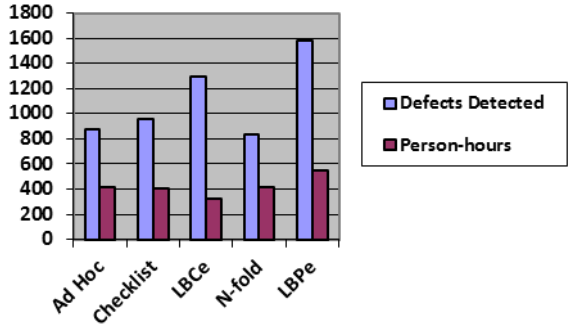

Fig. 7. Total de defeitos detectados e custo gasto em relação à técnica de leitura utilizada.

Detectando 1293 defeitos com a utilização da técnica de leitura baseada em cenários, observou-se que foram detectados 35% e 48% de defeitos a mais que as técnicas de leitura Checklist e Ad Hoc, respectivamente. Outra observação que pode ser notada ao analisar o gráfico é a superioridade da técnica LBPe em relação às demais, que segundo experimentos realizados no mundo real é a técnica mais eficiente entre todas.

5 CONCLUSÕES E TRABALHOS FUTUROS

Ao utilizar o modelo de dinâmica de sistemas descrito neste artigo, gerentes de projeto podem realizar simulações para obter um diagnóstico "a priori" sobre os possíveis impactos da realização de inspeções nos diferentes tipos de documentos de projetos de software. O modelo permite aos gerentes definir cenários de acordo com as características da empresa, do projeto e da equipe que se deseja simular, permitindo antever os impactos das decisões gerenciais acerca da maneira como será realizada a atividade de inspeção. As decisões e intervenções gerenciais podem ser testadas com a realização de simulações e os resultados obtidos permitem analisar o retorno obtido com a alocação de recursos para a atividade de inspeção.

Em empresas que já realizam inspeções durante o desenvolvimento de software, o modelo pode ser utilizado para analisar o impacto de alterações no modo como essa atividade é realizada. As simulações com o modelo permitem reduzir o tempo, o custo e o risco dos experimentos, que não precisam ser feitos em projetos reais. Diversos cenários podem ser testados até que seja obtido o mais adequado para a empresa, em termos de número de defeitos detectados e recursos alocados.

Com o objetivo de desenvolver um modelo capaz de produzir resultados cada vez mais próximos da realidade, os próximos passos desse trabalho são: melhorar a quantificação dos relacionamentos entre as variáveis presentes no modelo e integrar o modelo descrito nesse trabalho com os modelos produzidos por (AMBROSIO, 2011) e (HERMSDORF, 2011) para obter um modelo mais completo e robusto, abordando várias atividades do processo de desenvolvimento de software.

2.5 System dynamics model for simulation of the software inspection process

Jailton S. Coelho1, José Luis Braga1, Bernardo Giori Ambrósio

ABSTRACT

Repairing a defect in the late phases of software development can be a hundred times more expensive than finding and fixing it during the requirements and design phase. Software inspection is a technique that may be used to aid in the identification of defects during early stages of the process and avoid propagation of such defects to later phases. The cost-benefit of inspections may be significant if they are efficiently performed. Since this process is affected by several quality factors, the analysis of the overall context of inspection may become complex. Project managers are reluctant to introduce inspection due to uncertainty regarding its real benefits. This paper presents a system dynamics model, which is a descriptive technique for systems modeling and simulation and involves several variables that strongly influence inspection efficiency. The influence levels of model variables are quantified based on real or empirical experiments reported in the literature, in order to approximate model results to values that would be obtained in the real world. The model allows the reproduction of scenarios without paying the costs and facing the risks of a real project implementation. Therefore, it enables the analysis of inspection effects on the software development process.

Keywords: Software Inspection. System Dynamics. Defect Detection.

1 INTRODUCTION

Approximately 40 to 50% of the effort in software development is due to rework, mostly to remove defects, which sometimes demand significant changes in the architecture of the project (SHULL, 2002). The cost to remove a software defect increases as the process advances to late development phases. Finding and fixing a defect late in the process is often 100 times more expensive than detecting the problem in phases closer to that in which the defect was introduced (SHULL, 2002).

Laitenberger and Debaud (1991) consider a defect as any deviation from predefined quality properties, including the non-satisfaction of elicited requirements.

The quality of a software is an aspect to be monitored and adjusted throughout the development process (RUSSEL, 1991). Activities such as defect prevention and detection must be inserted throughout the process in order to decrease the costs of repairing a defect in later stages of development. Assigning resources to the review of software artifacts as soon as they are finished may reduce rework effort and improve product quality (FAGAN, 1986). Finding a defect as early as possible can provide a significant contribution to the project, in terms of both economy and quality.

Software inspection is a specific type of review that can be applied to software artifacts and presents a rigorous and well-defined process of defect detection (PORTER, 1995). According to Laitenberger and Debaud (LAITENBERGER, 1991), the primary objective of an inspection is to detect defects before the testing phase begins, which contributes to improve the overall quality of the software with respect to its budget and schedule. Despite these advantages, project managers are often reluctant to use inspection during software development, mostly because they question the efficiency of the results (FAGAN, 1986) due to the lack of tools that provide for an estimate of the return on investment resulting from assigning resources to such activities.

This paper presents a system dynamics model for the assessment of the impacts of assigning resources to inspection activities in several phases of the software development process. The model enables the simulation and analysis of results of management policies regarding inspections prior to their application into a real project. The model can be used as a tool for decision support since it enables the simulation of different scenarios according to characteristics of the company, project and team, without the need to perform experiments with real projects, which would be expensive and time-consuming.

The model included variables such as the number of team members, reading technique, company's Software Capability Maturity Model (SW-CMM) level, type and size of the document, among others, which directly influence the quality of the inspection process, as well as the investment return obtained from assigning resources to this activity. These variables and the existing influence relationships among them were identified and calibrated based on real or empirical experiments

reported in the literature, in order to ensure model consistency and coherence with the knowledge disseminated by Software Engineering.

The structure of the paper is as follows: Section 2 addresses the concepts in the context of this study and some related papers; Section 3 describes the process of model construction; Section 4 presents the simulations performed with the proposed model to assess the influence levels of the factors on inspection activity; finally, Section 5 shows our conclusions and recommendations for future research.

2 CONTEXT

This paper consists of an applied or technological research that utilizes the basic knowledge available in the literature to provide new knowledge for practical applications. This type of research is based on an observational study and, with respect to research procedures, is characterized by an extensive literature review (JUNG, 2004).

2.1 Software Inspection

Inspection is a particular review process that contributes to ensuring the quality of a software product. All phases of the software development process are susceptible to the introduction of defects, which can be detected by inspection and subsequently removed.

The basic process of inspection consists of six consecutive steps (KELLY, 1992):

- Planning. Performed by a moderator who has the responsibility to define the context of the inspection (inspection description, reading technique, document to be inspected, and author of the document, among others), select the inspectors and distribute material to the inspection team.
- Defect detection. The selected inspectors have the task to find possible defects in the documents they receive and produce a list of possible defects during the inspection.
- Defect collection. The moderator must collect the lists of defects made by the inspectors, eliminate repeated defects, i.e., those that were found by more than one inspector, and keep a single record of each defect.

- Defect discrimination. The moderator, inspectors and author of the document discuss the defects in order to classify them as an actual defect or as a false positive. False positives are eliminated from the list of defects, and only the real defects remain.
- Rework. Defects on the list are corrected by the author of the document and a correction report is produced at the end.
- Follow-up. The moderator decides if the author has corrected all defects or if a new inspection must be performed.

Inspectors may assume the following roles (LAITENBERGER, 1991):

- Organizer: plans all inspection activities within a project.
- Moderator: conducts the activities and controls the meetings.
- Author: develops the product to be inspected and removes the identified defects.
- Presenter: describes, explains and interprets the sections of the material.
- Recorder: classifies and registers all defects detected during the inspection meeting.
- Collector: collects the defects detected by the inspectors when there is no inspection meeting.
- Inspector: identifies defects in the document. All team members can act as inspectors, regardless of their other responsibilities.

Many developers have skills in writing requirement documents and software codes, but they are not offered any support such as an adequate review of these artifacts, which could be performed using reading techniques, for instance.

Reading techniques contribute to increase the comprehension of a software artifact to a level which enables the inspectors to identify the important information to understand the document, as well as how such information relates to the problem being addressed (FAGAN, 1986). Some of the main reading techniques are (LAITENBERGER, 1991):

- Ad Hoc: Ad Hoc reading does not offer any technical support on how to detect defects in a software artifact. In this case, defect detection depends exclusively on the skill, knowledge and previous experience of an inspector.
- Checklist (CBR): this technique is based on a series of questions (often yes or no questions) about subjects of the document to be inspected.

- Scenario (SBR): the scenario-based reading technique employs the notion of scenarios that provide personalized orientation to inspectors on how to detect defects. A scenario may be a set of questions or a more detailed description of how to perform the analysis of the document.
- Stepwise Abstraction (SA): this technique provides more elaborate and precise reading instructions for code artifacts. When reading a sequence of statements in the code, the inspector abstracts the function that these statements compute. This procedure is repeated until the last function of the code artifact inspected has been abstracted and compared with the specification.
- Function Point Analysis (FPA): the FPA defines a system in terms of its inputs, files, enquiries, and outputs, and the function point scenarios are developed around these items. A function point scenario consists of questions and directs the focus of an inspector to a specific function point item within the requirements document inspected.
- Perspective (PBR): the perspective-based reading technique is based on the idea that a software product must be inspected from the perspective of the different stakeholders. These perspectives are elaborated in order to force inspectors to see from the view points of the user, author and tester, using the corresponding perspective.
- N-fold: this method consists of the application of the formal inspection by N teams, which carry out parallel inspections. One moderator is responsible for coordinating and merging the results into one list of defects. This technique is applied considering the hypothesis that different teams improve the efficiency of the process by detecting more defects than a single inspection team

2.2 System Dynamics

System Dynamics was developed in the 1950s by the electrical engineer Jay Forrester, who published in 1961 the book Industrial Dynamics. However, it was only around 1968 that this subject gained strength with famous models of urban and world strategic studies, disseminated in the books Urban Dynamics and World Dynamics (FORRESTER, 1980).

System dynamics is a descriptive technique used for system modeling and simulations. This technique is based on the systemic thinking and analysis and on the mathematical theory of dynamic systems. System Dynamics enables the comprehension of how the adopted policies or even the structure of the system affect or determine its dynamic behavior. Thus it is possible to forecast future behaviors and eventual collapses. The use of such techniques allows the analysis and understanding of problems and situations in an integrated and interconnected manner, showing the existing relationships among decision variables (AMBROSIO, 2011).

A system dynamics model consists of two main elements: stocks that accumulate resources of the system, and flows or rates, which are functions that represent the decisions or policies of the company regarding the use or accumulation of stocks or resources. There are also the converters/simple variables, which are elements that influence the flow values (MADACHY, 1996). These elements are shown in Figure 1.

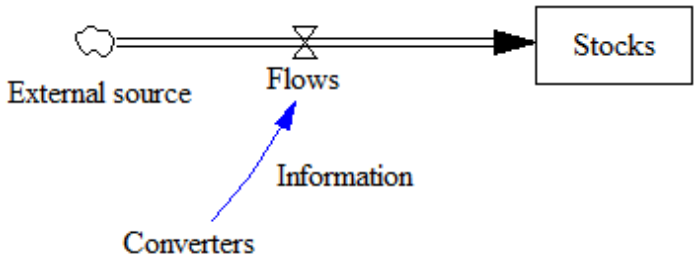

Figure 1. Basic elements of a generic stock and flow model (MADACHY, 1996).

The elements of Figure 1 are:

- Stocks: represent the accumulations or decrements over time. These values are influenced by input and/or output flows.
- External source: comprises the infinite resources or repositories used by the flows that are not specified in the model and not included in the simulation context.
- Flows: may be considered as “actions” in the system. They directly influence the stocks and may represent the policies and decisions adopted by the company.
- Converters (or auxiliary variables): used for the detailed design of the model structure to represent the variables that influence the flows. Their relationships with the flows are represented by unidirectional arrows.

- Information (unidirectional arrows): represent information flows and interconnect model variables. These arrows allow the construction of feedback loops in the model (STERMAN, 2000).
- Constants: represent the variables that their values not change during the simulation.

2.3 Related Work

One of the most widespread system dynamics models in Software Engineering was proposed by Abdel-Hamid (1990) to simulate software development projects. This model takes into account the following aspects of a project: Human Resource Management, Software Production, Software Development, Quality Assurance and Rework, Tests and Control.

Madachy (1996) used System Dynamics to model the effects of formal inspections on several stages of the software development life cycle. Impacts on cost, schedule and quality were compared for different inspection rates. The model also addressed the interrelated flows of tasks, errors and human resources throughout the different phases of software development projects.

The model we propose in this study also uses system dynamics. However, our model simplifies the view of the factors that influence the inspection quality and efficiency, allows the users to interact with the model and is focused on defect detection, which is one of the six phases of the inspection activity.

3 MODEL CONSTRUCTION

The system dynamics model presented in this study was constructed based on the method proposed by Ambrósio (2011), which consists of three phases subdivided into one or more steps. In the first phase, we identify the variables involved in the process or problem to be modeled. Subsequently, the relationships among these variables were identified with the aid of influence diagrams that allow the construction of simple models for easy understanding. These diagrams allow the modeling of the influence relationships among variables and facilitate the identification of feedback loops (STERMAN, 2000). In the second phase, the influence diagrams were mapped into system dynamics models according to some rules established in the proposed method. In the third phase, the system dynamics

models were refined and the relationships among their elements were quantified and tested until a final and validated model was obtained.

3.1 Variable Selection

Initially, we identified the variables (Figure 2) that represent the factors that mostly influence the inspection of software project artifacts. According to Laitenberger and Debaud (1991), these variables are team characteristics, effort, reading technique, organization of defect detection activity and product characteristics.

Team characteristics are related to the number and experience of the inspectors involved in the process. The effort represents the total cost required for the inspection. The reading technique, which may be different for each life-cycle product, can be defined as the procedures or mechanisms for inspectors to detect defects in the inspected product. The organization of the defect detection activity is related to the number of inspections to be performed, reading technique and number of inspectors on the team. The product characteristics are represented by the life-cycle, size, difficulty and initial quality of the product. The life cycle refers to the project development phase. The size of the document regards the number of pages. The difficulty of the product refers to its complexity, i.e., the more complex the document, the more difficult it is to detect a defect. The initial quality of the products refers to the number of defects that can possibly be present in the software artifact, i.e., the higher the quality, the lower the number of existing defects

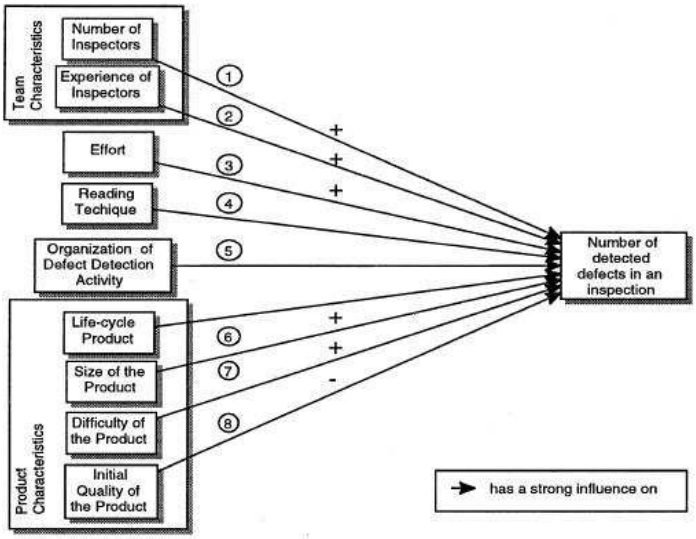

Figure 2. Factors that strongly influence the performance of the inspection process (LAITENBERGER, 1991).

3.2 System Dynamics Model

After the identification of the relationships among variables based on data extracted from literature references cited throughout this paper, influence diagrams were constructed to represent the cause and effect relationships between variables. Subsequently, these influence diagrams were mapped into a system dynamics model according to the following rules:

- Variables that represent something that increases or decreases over time were mapped as stocks;
- Variables that directly influence the stocks were mapped as flows;
- Variables that represent input parameters for model simulation were turned into constant converters, i.e., their values do not change during the simulation process;
- Variables that were not classified into any of the previous categories (stocks, flows and constants) were mapped as converters.

In order to construct the system dynamics model we used the free version of the Vensim (2010) tool for academic use. The model obtained is presented in Figure 3.

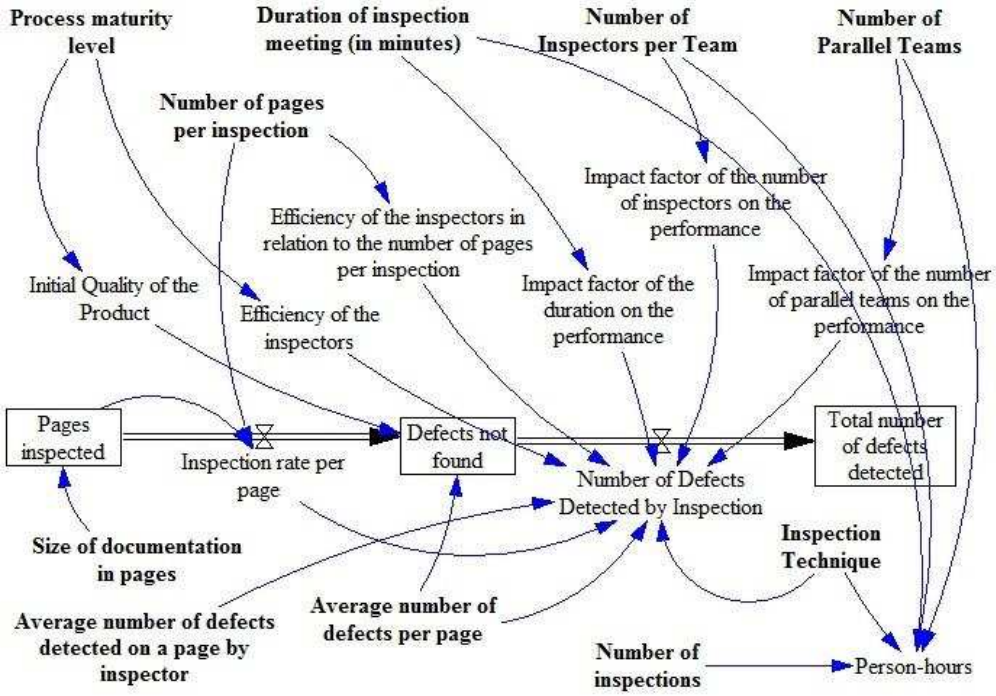

Figure 3. System dynamics model of stocks and flows.

The variable Process Maturity Level represents the company's CMM level. It positively influences the variables Efficiency of the Inspectors and Initial Quality of the Product, which account for the productivity of the inspectors and the initial quality of the artifact to be inspected, respectively. The influence relationship between the variable Process Maturity Level and the variables Efficiency of the Inspectors and Initial Quality of the Product was quantified based on studies carried out by Clarck (2000). By varying the company's CMM level from the lowest to the highest value, the efficiency of the inspectors can increase by up to 43%, whereas the number of defects contained in the document can be reduced by 62%.

The Number of Pages per Inspection represents the number of pages of the document allocated to each inspection. This variable positively influences the Inspection Rate per Page, and negatively influences the Efficiency of the Inspectors in Relation to the Number of Pages per Inspection. The influence of the variable Number of Pages per Inspection on the efficiency of the inspectors was modeled as a graphic variable based on experiments carried out by Kelly *et al*. (1992).

A graphic variable is used when the mathematical equation that defines the relationship between two variables is unknown. This variable enables the quantification of the relationship between such variables through the sketch of a graph that represents this mathematical function.

The variable Duration of the Inspection Meeting not only affects the number of defects detected by inspection but also the costs. The variable Impact Factor of the Duration on the Performance is a graphic variable that corresponds to a mathematical function that quantifies the influence relationship between the duration of the inspection meeting and the number of defects detected by inspection. The sketch was created based on studies carried out by Dunsmore *et al*. (2000) who recommends a maximum of 2 hours to complete the inspection.

The variable Number of Inspectors per Team represents the size of the team responsible for the defect detection activity. The influence on the number of defects detected by inspection is defined by the graphic variable Impact Factor of the Number of Inspectors on the Performance. The sketch of the graph was created based on empirical experiments carried out by Walia and Carver (2008), who simulated the inspection process with 73 inspectors.

The variable Number of Parallel Teams represents the number of parallel teams responsible for the N-fold reading technique. The relationship between the number of

parallel teams and the number of defects detected by inspection is quantified by the graphic variable Impact Factor of the Number of Parallel Teams on the Performance, which was modeled based on experiments carried out by Kantorowititz *et al*. (1997), who simulated the inspection process with up to 10 parallel teams, increasing the number of detected defects by up to 100%.

The size of the document to be simulated is represented by the variable Size of Documentation in Pages, which can assume a maximum value of 60 thousand pages. This limit was defined according to studies carried out by Clarck (2000) who refers to source code documents of the same size as that adopted in the model proposed in this study.

The variable Number of Inspections represents the number of inspections to be performed in the document. The reading technique used in the simulations is determined by the variable Inspection Technique. The variable Person-hours represents the effort required to perform the inspections. The variables Average Number of Defects per Page, Average Number of Defects Detected on a Page by Inspector, The Number of Defects Detected by Inspection and The Inspection Rate per Page are obvious.

The Total Number of Defects Detected stock represents the sum of defects detected during the inspections and the Defects Not Found stock represents the number of defects not detected by the inspections. The Pages Inspected stock represents the number of pages of the document to be inspected.

3.3 Control Panel

The control panel is a tool that allows the users to interact with the model and perform adjustments in order to set scenarios to be simulated. For the model described in this study, we defined the control panel presented in Figure 4. It allows project managers to adjust the values for model input variables according to the specific characteristics of the organization, team and type of inspection, which define the context of the scenario to be simulated.

Each model input variable may assume values within a continuous range, and has a minimum and a maximum value. The limits maximum e minimum was defined according the respective font that quantifies the influence relationship between the variables. Variable values may be altered by typing in values or using scroll bars. If

the user enters a non-valid value for a certain variable, it will change to the closest valid value. When updating the control panel, the user must be careful not to input an incompatible combination of values for the variables, to avoid destabilization of the model and inconsistent results. Decision makers must be familiar with the problem and the acceptable combinations of values for the variables.

The range of permitted values for some of the input variables in the model's control panel is represented by a bar whose color gradually changes from green to red. The values closer to the green color are within the expected range described in the literature, whereas the values closer to red can lead to an unstable model. For the variables Average Number of Defects per Page and Average Number of Defects Detected on a Page by Inspector, the values closer to green represent the mean values obtained in experiments reported in the literature considering each type of document inspected. As for the variables Duration of the Inspection Meeting (in minutes), Number of Inspectors per Team, Number of Pages per Inspection and Number of Parallel Teams, the values closer to green are recommended by several experiments reported in the literature, i.e., when setting values outside that specific range, project managers will be simulating scenarios that represent situations where resources would be wasted in inspections. The variable Inspection Technique represents the five reading techniques that can be simulated: Ad Hoc, CBR, PBR, SBR and N-fold. Finally, the variable Process Maturity Level represents the 5 CMM levels (CLARCK, 2000) that can be set in the simulations.

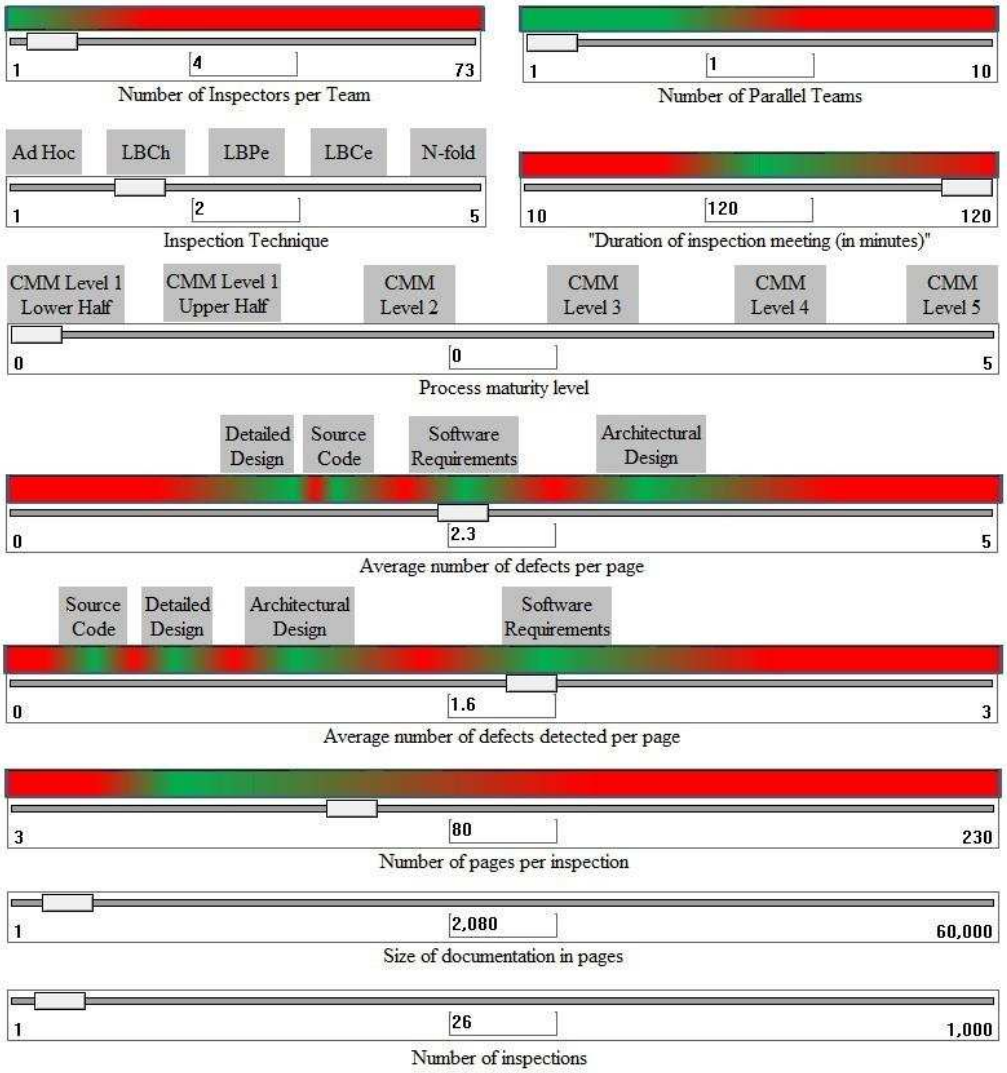

Figure 4. Control panel for the configuration of scenarios.

4 MODEL SIMULATION, VALIDATION AND USE FOR DECISION SUPPORT

The following sections describe the simulation results for the developed system dynamics model. A baseline scenario was defined and simulated first. Subsequently, after the alteration of values defined for the baseline scenarios, other simulations were performed. The results obtained allowed us to assess the impacts of model input variables on the number of defects detected and inspections costs. The values assumed by the variables in the baseline scenario were based on experiments conducted by Kelly *et al*. (1992) and Clarck (2000). These values are shown in Table 1 and represent a situation in the real world.

Table 1. Values assumed for the variables in the baseline scenario.

Variable	Value
Number of pages	2000
Average number of defects per page	1.6
Average number of defects detected by inspector	0.2
Number of pages per inspection	40
Number of inspections	50
Process maturity level	CMM level 1 Lower Half
Inspection Technique	CBR
Duration of the inspection meeting	120 minutes
Number of parallel teams	1
Number of inspectors per team	4

4.1 Influence of the Number of Parallel Teams

Additional inspection teams can significantly increase the number of defects detected by inspection. The primary use of N-Fold inspection is to identify faults that might not be detected by a single inspection team (KANTOROWITITZ, 1997). Figure 5 shows the simulation results considering different numbers of parallel teams, which allows us to analyze the impact of additional teams on the number of defects detected and costs.

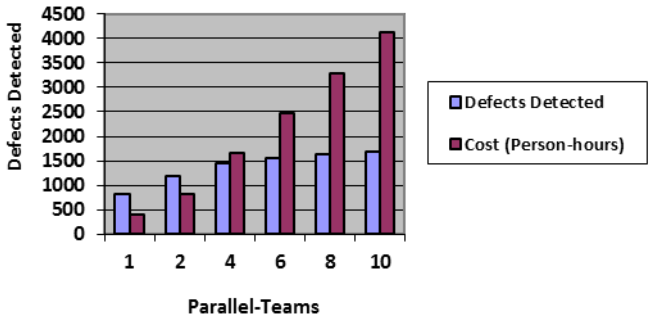

Figure 5. Number of defects detected in relation to the number of parallel teams.

As observed in Figure 5, a single inspection team results in the detection of 833 defects at a cost of 412 person-hours. However, the use of 10 parallel teams resulted in 1674 defects detected and required 4120 person-hours. Thus an increase in the number of parallel teams does not result in a proportional increase in the number of

defects detected. Therefore, the choice of the most suitable number of teams depends on the available resources and their familiarity with the problem.

4.2 Influence of the Size of the Inspection Team

Similar to the variable Number of Parallel Teams, the size of the inspection team may also significantly influence the number of defects detected and the costs. However, the addition of one inspector to the team increases the number of defects detected by inspection as well as decreases the amount of time required for the inspection, since part of the document is allocated to the new inspector. The results obtained by changing the number of inspectors are shown in Figure 6.

Figure 6. Number of defects detected in relation to the number of inspectors.

As shown in Figure 6, a team of 3 inspectors detected 896 defects at a cost of 300 person-hours. A team of 10 inspectors detected 1098 defects and the inspection process required 1000 person-hours. The increase in the number of defects detected by a team of 20 inspectors is relatively low, whereas the costs significantly increase. Thus the most suitable number of inspectors to be used in the defect detection activity depends on the available resources, taking into account the limit after which the results no longer improve.

A consensus has not been reached on the ideal size of the inspection team, but some recommendations are reported in the literature. Madachy (1996) suggests between 3 and 5 reviewers, whereas Kantorowititz *et al*. (2010) recommends a number between 3 and 4 inspectors per team.

4.3 Influence of the Number of Pages per Inspection

The Number of Pages per Inspection influences the number of defects detected by inspection and costs, and also determines the time required for the inspection. As the number of inspections increases, the number of pages per inspection decreases. The results obtained by changing the number of pages per inspection are shown in Figure 7.

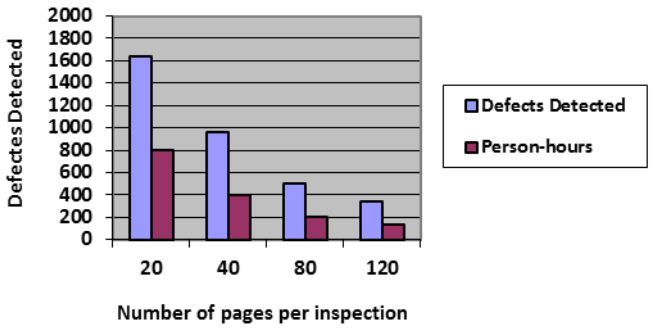

Figure 7. Number of defects detected in relation to the number of pages per inspection.

For 80 pages per inspection, 506 defects were detected, which required 200 person-hours. However, when the number of pages per inspection was limited to 20, 1634 defects were detected at a cost of 800 person-hours. As observed in Figure 7, there is a significant increase in the number of defects detected with the decrease in the number of pages per inspection. Kelly *et al*. (1992) states that for a team of 6 inspectors, and a duration of inspection meeting limited to 2 hours, the ideal number of pages to be covered is 40 pages per inspection.

4.4 Duration of the Defect Detection Activity

In the model proposed in this study, the duration of the defect detection activity may reach up to 2 hours. However, the results presented in Figure 8 for different durations of the inspection activity show that after 1 hour of inspection, the increase in the number of defects detected by inspection is relatively low.

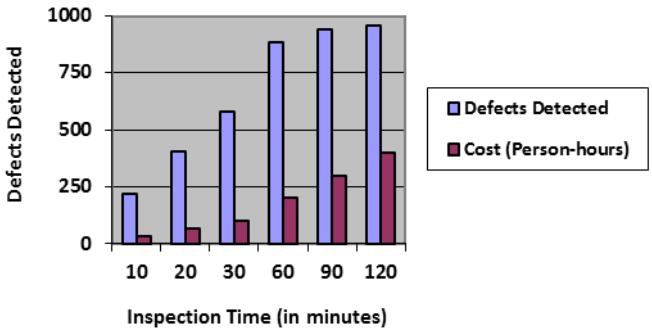

Figure 8. Number of defects detected in relation to inspection time.

By limiting inspection time to 60 min, a number of 884 defects were detected at a cost of 200 person-hours. However, when inspection time increased to 120 min, 957 defects were detected and 400 person-hours were required. Thus for a 2-hour inspection, costs were increased by 100%, but the number of defects detected increased only 8.4%.

4.5 Influence of the Reading Technique

According to Basili (1996), inspections of requirement documents using perspective-based reading techniques would result in 35% more defects detected compared with the Ad Hoc reading. On the other hand, the author recognizes that the application of such technique requires 30% more effort. This study clearly shows that each technique used in software inspection has different efficiency and effort levels. Therefore, the reading technique with the highest indices of defect detection may not always be the best choice for the project. The simulation results obtained from using different reading techniques are shown in Figure 9.

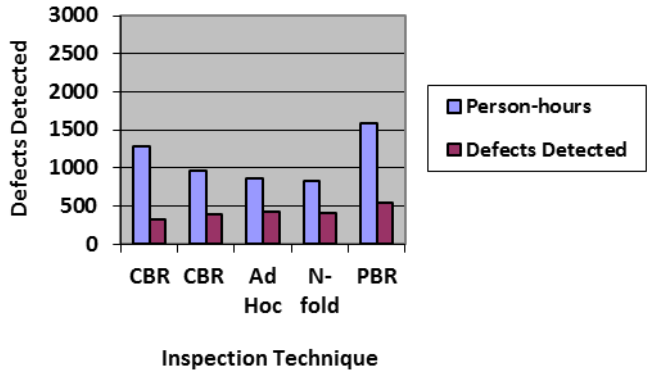

Figure 9. Number of defects detected in relation to the reading technique applied.

The scenario-based reading technique resulted in the detection of 1293 defects, which represents increases by 35% and 48% in the number of defects detected using the CBR and Ad Hoc reading, respectively. Another observation that can be drawn from Figure 9 is the superiority of the PBR technique in comparison with the others. According to experiments carried out in the real world, this is considered the most efficient technique.

4.6 Using the Model for Decision Support

Model simulation considering the baseline scenario configuration resulted in a total of 958 defects detected, i.e., 30% of the total estimated number of defects in the document, at a cost of 400 person-hours.

In order to show the model's application for decision support, we defined the objective to increase the number of defects detected to 40% of the total estimated number of defects, at the lowest possible cost. This can be achieved by changing the input parameter values simultaneously, but for simplification we altered one parameter at a time. The parameters analyzed were the number of parallel teams, number of inspectors, number of pages per inspection and reading technique. The results are shown in Table 2.

Table 2. Cost-benefit relationship of parameters obtained from changing the baseline scenario.

Variable	Baseline scenario	Simulated scenario	Number of defects detected	Cost (Person-hours)	Percentage of defects detected
Parallel teams	1	3	1347	1236	42.1%
Number of inspectors	4	73	1204	7300	37.6%
Number of pager per inspector	40	37	1297	592	40.5%
Reading technique	CBR	SBR	1293	323	40.4%

Data presented in Table 2 show that the best decision for achieving the proposed goal is to use the SBR reading technique, which detected 40.4% of the

defects in the document and required only 323 person-hours. Also, if the only change in the baseline scenario was the addition of new inspectors, the proposed objective would not be achieved.

5 CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

The system dynamics model described in this study enables project managers to perform simulations and obtain a prior diagnosis of the expected impacts of inspections on different types of documents of software projects. The model allows managers to define scenarios according to the characteristics of the team of inspectors, software project and organization under analysis, and thus foresee the impacts of management decisions related to inspection activities. Management decisions and interventions may be tested with simulations and the results allow the analysis of the return obtained from the allocation of resources to the inspection activity.

Organizations that already carry out inspections during software development can use the model to analyze the impact of changes in the procedures used in the activity. The model simulations estimate the reduction of time, costs and risks of the experiments, which do not have to be implemented in real projects. Several scenarios can be tested until the most suitable choice in terms of number of defects detected and resources allocated is obtained for the organization.

The model enables the estimation of the total number of defects in the product, as well as the effectiveness and cost of the inspection activity, which are used by some organizations as criteria to decide on the continuation of inspections.

With the objective of developing a model consistent with reality, further research on this subject should focus on improving the quantification of the relationships among model variables and integrating the model developed in this study with those constructed by Ambrósio (2011) and Hermsdorf (2011), to obtain a more complete and robust model that addresses activities in the process of software development.

ACKNOWLEDGMENTS

We would like to thank CAPES for financial support to this research.

3 CONCLUSÕES

A partir dos estudos e pesquisas realizadas, foi construído um modelo de dinâmica de sistemas para simulação do processo de inspeção de software. A discussão acerca dos relacionamentos e das interações entre as variáveis envolvidas, explicitados no modelo, contribuem para aumentar a compreensão acerca dos aspectos gerenciais que afetam o processo de inspeção. Com isso, o objetivo geral, proposto nessa pesquisa, foi atingido.

Para usar o modelo como um veículo de experimentação, foi definido um painel de controle contendo algumas variáveis que permitem ajustar o modelo ao realizar as simulações. As variáveis de entrada permitem configurar o modelo de acordo com as características da organização, da equipe e do projeto que definem o contexto da simulação.

O modelo permite realizar simular cenários que são inviáveis de serem feitos em ambientes reais devido ao custo e ao tempo necessários. As simulações possibilitam aos gerentes verificar os impactos da materialização de riscos e as consequências dinâmicas das decisões gerenciais planejadas, antes que elas sejam implementadas em um projeto real.

A capacidade de configurar o modelo, por meio de um painel de controle, torna amigável a interação do usuário com o modelo, facilitando a configuração de novos cenários. O modelo permite estudar e antever as interações entre as variáveis mediante os cenários, e auxilia na descoberta da ação gerencial mais adequada para lidar com um determinado problema.

As análises gerenciais suportadas pelo modelo consistem em verificar:

- os impactos do tamanho da equipe de inspetores;
- os impactos da técnica de inspeção utilizada;
- os efeitos do nível CMMI da empresa em questão;
- os efeitos da duração da atividade de inspeção;
- os efeitos da definição do número de páginas que serão alocadas por inspeção.

Os relacionamentos entre as variáveis do modelo foram definidos e quantificados a partir de informações publicadas na literatura. Isso torna a estrutura do modelo embasada no conhecimento disponibilizado pela comunidade científica, garantindo uma maior confiabilidade aos resultados obtidos durante as simulações.

O modelo foi construído por meio de diversos refinamentos. Cada versão obtida do modelo proporcionava um maior entendimento sobre os relacionamentos entre as variáveis, provocando modificações no modelo e originando uma nova versão. Ao utilizar a dinâmica de sistemas para modelar um problema ou sistema, o próprio processo de construção do modelo gera um processo de aprendizagem que gera uma nova construção e assim sucessivamente, constituindo um ciclo de reforço positivo.

3.1 Trabalhos futuros

O modelo construído não abrange todos os fatores envolvidos no processo de inspeção. Uma possível extensão para esse trabalho é a realização de novos refinamentos no modelo com o objetivo de incluir novas variáveis e relacionamentos, oferecendo suporte a novos cenários. Também podem ser construídos novos modelos ou integrar o modelo descrito nesse trabalho com os modelos produzidos por Ambrósio (2011) e Hermsdorf (2011) para obter um modelo mais completo e robusto, abordando várias atividades do processo de desenvolvimento de software.

APÊNDICE A

A.1 Equações das variáveis do modelo

(01) Average number of defects detected on a page by inspector = [0, 3]

Units: **undefined**

Fonte: (KELLY, 1992).

(02) Average number of defects per page = [0, 5]

Units: **undefined**

Fonte: (KELLY, 1992).

(03) Defects not found = INTEG (MAX (Inspection rate per pages * Average number of defects per page * Initial Quality of the Product - Number of Defects Detected by Inspection, 0), 0)

Units: **undefined**

(04) "Duration of inspection meeting (in minutes)" = [10, 120]

Units: **undefined**

Fonte: (DUNSMORE, 2000).

(05) Efficiency of the inspectors =

IF THEN ELSE (Process maturity level = 5, 1.448,

IF THEN ELSE (Process maturity level = 4, 1.422,

IF THEN ELSE (Process maturity level = 3, 1.363,

IF THEN ELSE (Process maturity level = 2, 1.286, 1))))

Units: **undefined**

Fonte: (CLARCK, 2000).

(06) Efficiency of the inspectors in relation to the number of pages per inspection = WITH LOOKUP (Number of pages per inspection, ([(1, 0) - 230, 6)], (1, 5.958), (3, 4.706), (5, 4.041), (7, 3.529), (9, 2.941), (13, 2.353), (16, 2.059), (19, 1.765), (24, 1.471), (28, 1.318), (32, 1.176), (40, 1), (51, 0.806), (71, 0.588), (86, 0.49), (115, 0.371), (144, 0.28), (230, 0.194)))

Units: **undefined**

Fonte: (KELLY, 1992).

(07) FINAL TIME = [1, ∞)

Units: Session

The final time for the simulation.

(08) Impact factor of the duration on the performance = WITH LOOKUP ("Duration of inspection meeting (in minutes)", [(0, 0) - (120, 1)], (0, 0), (10, 0.231), (20, 0.423), (30, 0.608), (40, 0.762), (50, 0.862), (60, 0.923), (70, 0.954), (80, 0.973), (90, 0.981), (100, 0.987), (110, 0.992), (120, 1)))

Units: **undefined**

Fonte: (DUNSMORE, 2000).

(09) Impact factor of the number of inspectors on the performance = WITH LOOKUP (Number of Inspectors per Team, ([(1, 1) - (73, 2.739)],(1, 1),(2, 1.743), (3, 2.04), (4, 2.179), (5, 2.268), (6, 2.336), (7, 2.381), (8, 2.427), (9, 2.468), (11, 2.527), (13, 2.564), (19, 2.632), (28, 2.675), (73, 2.739)))

Units: **undefined**

Fonte: (WALIA, 2008).

(10) Impact factor of the number of parallel teams on the performance =

IF THEN ELSE (Number of Parallel Teams = 1, 1,

IF THEN ELSE (Number of Parallel Teams = 2, 1.411,

IF THEN ELSE (Number of Parallel Teams = 3, 1.616,

IF THEN ELSE (Number of Parallel Teams = 4, 1.74,

IF THEN ELSE (Number of Parallel Teams = 5, 1.822,

IF THEN ELSE (Number of Parallel Teams = 6, 1.88,

IF THEN ELSE (Number of Parallel Teams = 7, 1.924,

IF THEN ELSE (Number of Parallel Teams = 8, 1.96,

IF THEN ELSE (Number of Parallel Teams = 9, 1.987, 2.009)))))))))

Units: **undefined**

Fonte: (KANTOROWITITZ, 1997).

(11) Initial Quality of the Product =

IF THEN ELSE (Process maturity level = 1, 1,

IF THEN ELSE (Process maturity level = 2, 0.63,

IF THEN ELSE (Process maturity level = 3, 0.5,

IF THEN ELSE (Process maturity level = 4, 0.427, 0.38))))

Units: **undefined**

Fonte: (CLARCK, 2000).

(12) INITIAL TIME = 0

Units: Session

The initial time for the simulation.

(13) Inspection rate per pages =

IF THEN ELSE (Pages inspected > Number of pages per inspection, Number of pages per inspection,

IF THEN ELSE (Pages inspected > 0, Pages inspected, 0))

Units: **undefined**

(14) Inspection Technique = [Checklist, Ad Hoc, LBPe, LBCe, N-Fold]

Units: **undefined**

Fonte: (BERTINI, 2006; CIOLKOWSKI, 2003; KANTOROWITITZ, 1997; PORTER, 1995).

(15) Number of Defects Detected by Inspection =

IF THEN ELSE (Inspection Technique = 1, MIN (Impact factor of the number of inspectors on the performance * Inspection rate per pages * Average number of defects detected on a page by inspector * 1 * Impact factor of the duration on the performance * Efficiency of the inspectors * Efficiency of the inspectors in relation to the number of pages per inspection, Average number of defects per page * Inspection rate per pages),

IF THEN ELSE (Inspection Technique = 2, MIN (Impact factor of the number of inspectors on the performance * Inspection rate per pages * Average number of defects detected on a page by inspector * 1.099 * Impact factor of the duration on the performance * Efficiency of the inspectors * Efficiency of the inspectors in relation to the number of pages per inspection, Average number of defects per page * Inspection rate per pages),

IF THEN ELSE (Inspection Technique = 3, MIN (Impact factor of the number of inspectors on the performance * Inspection rate per pages * Average number of defects detected on a page by inspector * 1.819 * Impact factor of the duration on the performance * Efficiency of the inspectors * Efficiency of the

inspectors in relation to the number of pages per inspection, Average number of defects per page * Inspection rate per pages),

IF THEN ELSE(Inspection Technique = 4, MIN (Impact factor of the number of inspectors on the performance * Inspection rate per pages * Average number of defects detected on a page by inspector * 1.484 * Impact factor of the duration on the performance * Efficiency of the inspectors * Efficiency of the inspectors in relation to the number of pages per inspection, Average number of defects per page * Inspection rate per pages), MIN (Impact factor of the number of inspectors on the performance * Inspection rate per pages * Average number of defects detected on a page by inspector * 0.956 * Impact factor of the number of parallel teams on the performance * Impact factor of the duration on the performance * Efficiency of the inspectors * Efficiency of the inspectors in relation to the number of pages per inspection, Average number of defects per page * Inspection rate per pages)))))

Units: **undefined**

Fonte: (LAITENBERGER, 2000).

(16) Number of inspections = FINAL TIME

Units: **undefined**

(17) Number of Inspectors per Team = [1, 73]

Units: **undefined**

Fonte: (WALIA, 2008).

(18) Number of pages per inspection = [3, 230]

Units: **undefined**

Fonte: (KELLY, 1992).

(19) Number of Parallel Teams = [1, 10]

Units: **undefined**

Fonte: (KANTOROWITITZ, 1997).

(20) Pages inspected = INTEG (- Inspection rate per pages, Size of documentation in pages)

Units: **undefined**

(21) "Person-hours" =

IF THEN ELSE (Inspection Technique = 1, 1.05 * Number of inspections * Number of Inspectors per Team * ("Duration of inspection meeting (in minutes)" / 60),

IF THEN ELSE (Inspection Technique = 2, Number of inspections * Number of Inspectors per Team*("Duration of inspection meeting (in minutes)" / 60),

IF THEN ELSE (Inspection Technique = 3, 1.365 * Number of inspections * Number of Inspectors per Team * ("Duration of inspection meeting (in minutes)" / 60),

IF THEN ELSE (Inspection Technique = 4, 0.809 * Number of inspections * Number of Inspectors per Team * ("Duration of inspection meeting (in minutes)" / 60), 1.03 * Number of inspections * Number of Inspectors per Team * ("Duration of inspection meeting (in minutes)" / 60) * Number of Parallel Teams))))

Units: **undefined**

(22) Process maturity level = [CMMI Level 1, CMMI Level 2, CMMI Level 3, CMMI Level 4, CMMI Level 5]

Units: **undefined**

Fonte: (CLARCK, 2000).

(23) SAVEPER = TIME STEP

Units: Session [0,?]

The frequency with which output is stored.

(24) Size of documentation in pages = [1, 60000]

Units: **undefined**

Fonte: (CLARCK, 2000).

(25) TIME STEP = 1

Units: Session [0, ?]

The time step for the simulation.

(26) Total number of defects detected = INTEG (Number of Defects Detected by Inspection, 0)

Units: **undefined**

REFERÊNCIAS BIBLIOGRÁFICAS

ABDEL-HAMID, T. K. Investigating the cost/schedule trade-off in software development. IEEE Software, jan., 1990.

AMBRÓSIO, B. G.; BRAGA, J. L.; RESENDE FILHO, M. A. Modeling and scenario simulation for decision support in management of requirements activities in software projects. Journal of Software Maintenance and Evolution (Print), v.23, p.35-50, 2011. http://dx.doi.org/10.1002/smr.469.

AURUM, A., PETERSSON, H. AND WOHLIN, C., "State-of-the-Art: Software Inspections after 25 Years", Software Testing, Verification and Reliability, 12(3):133-154, 2002.

BARTIÉ, A. Garantia de Qualidade de Software. Edição: 2002, Campus.

BASILI, V. R. *et al*. The Empirical Investigation of Perspective-Based Reading. Journal of Empirical Software Engineering, v. 1, n. 2, 1996.

BASTOS, Alexandre Antunes Parreiras. A Dinâmica de Sistemas e a compreensão de estruturas de negócios. São Paulo, SP. 2003.

BERTINI, L. A. Técnicas de Inspeção Aplicadas à Avaliação de Requisitos de Sistemas de Software: Um Estudo Comparativo. Piracicaba, 2006.

BLASCHEK, José Roberto. Gerencia de Requisitos: o principal problema dos projetos de software. Coluna produzida para a Developer's Magazine (ISLIG-RIO/PMI-ISSIG) URL: http://www.bfpug.com.br/islig-rio/Downloads/Ger%C3%AAncia%20de%20Requisitos-o%20Principal%20Problema%20dos%20Projetos%20de%20SW.pdf.

BOEHM, B.; BASILI, V. Software Defect Reduction Top 10 List. IEEE Software, vol. 34, nº 1, January 2001, p.135-137.

BRAGA, J. L.; SILVA, C. A. B.; WIAZOWSKI, B. A. E AVELLAR, S. O. C. (2004) “Modelagem com dinâmica de sistemas”. In: Santos M. L. e Vieira W. Métodos Quantitativos em Economia. Viçosa, MG: UFV, 2004. P. 411-434.

BRATLEY, P.; FOX, B. L.; SCHRAGE, L. E. A Guide to Simulation. New York: Springer-Verlag, 1987.

CIOLKOWSKI, M., LAITENBERGER, O., BIFFL, S., 2003, “Software Reviews: The State of the Practice”, IEEE Software 20 (6): 46-51.

CLARCK, B. Quantifying the Effects of Process Improvement on Effort. IEEE Software, v. 17, n. 6, p. 65-70, 2000.

DAIBERT, M. S. Monitoramento De Riscos Em Projetos De Software: Uma Abordagem Baseada Em Dinâmica De Sistemas E Técnicas De Inteligência Computacional. 2010.

DAVIS, A., DIESTE, O., HICKEY, A., JURISTO, N., E MORENO, A. M. (2006) "Effectiveness of Requirements Elicitation Techniques: Empirical Results derived from a Systematic Review". Proceedings of the 14th IEEE International Conference on Requirements Engineering, Minneapolis, USA. Setembro, 2006, pp. 179-188.

DIESTE, O. E JURISTO, N. (2010) "Systematic Review and Aggregation of Empirical Studies on Elicitation Techniques". IEEE Transactions on Software Engineering, 11 Fev. 2010. IEEE computer Society Digital Library. IEEE Computer Society.

DUNSMORE, A.; ROPER, M.; WOOD M. Object oriented inspection in the face of delocalization. Proceedings of the 22nd International Conference on Software Engineering, (Limerick), 2000.

GIBSON, D.; GOLDENSON , D.; KOST K. "Performance Results of CMMI-Based Process Improvement". Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2006-TR-004, 2006.

FAGAN, M. E. Advances in Software Inspections, IEEE Transactions of Software Engineering, v. SE-12, n. 7, p. 744-751, 1986.

FORRESTER, J. W.; SENGE, P. Tests for building confidence in system dynamics models. A. Legasto *et al*. (eds.), TIMS Studies in the Management Sciences (System Dynamics), North-Holland, The Netherlands, p. 209-228, 1980.

FRANZ, L. A.; SHIH, J. C. Estimating the value of inspections and early testing for software projects. CS-TR-6 Hewlett-Packard J, 1994.

HALL, A., "Seven Myths of Formal Methods", IEEE Software, setembro de 1990, pp. 11-20.

HERMSDORF, V. O. *et al*. Modelagem da atividade de elicitação de requisitos utilizando a técnica de entrevista: uma abordagem utilizando dinâmica de sistemas. XIV WER - Workshop em Engenharia de Requisitos Proceedings XIV CibSE. Rio de Janeiro, RJ, v. 14. p. 309-320, 2011.

ISRAR, A.; SHAHID, N. Minimizing Defects Originating from Elicitation, Analysis and Negotiation (E and A&N) Phase in Bespoke Requirements Engineering. Master Thesis Software Engineering. School of Engineering Blekinge Institute of Technology, Sweden. November, 2009.

JUNG, C. F. Metodologia para Pesquisa & Desenvolvimento. Rio de Janeiro: Axcel Books, 2004.

KALINOWSKI, M.; SPÍNOLA, R. O.; TRAVASSOS, G. H. Infra-estrutura Computacional para Apoio ao Processo de Inspeção de Software Resolução de Sistemas Lineares com Alta Exatidão no Ambiente de Agregados. Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ, Rio de Janeiro, RJ, 2004.

KANTOROWITITZ, E.; GUTTMAN, A.; ARZI, L. The Performance of the N-Fold Requirement Inspection Method. Requirements Engineering, v.2., n. 3, p. 152-164, 1997.

KELLY, J. C.; SHEIRIF, J. S.; HOPS, J. An analysis of defect densities found during software inspections. Journal of Systems Software, v.17, n. 2, p. 111-117, 1992.

KNIGHT, J. C.; MYERS, E. A. An Improved Inspection Technique. Communications of the ACM, v. 36, n. 11, p. 51-61, 1993.

LAITENBERGER, O.; DEBAUD, J. An encompassing life cycle centric survey of software inspection. Journal of Systems and Software, v. 50, n. 1, p. 5-31, 2000.

LAMSWEERDE, A. v. 2000. Formal specification: a roadmap. In Proceedings of the Conference on the Future of Software Engineering (Limerick, Ireland, June 04 - 11, 2000). ICSE '00. ACM, New York, NY, 147-159. DOI=http://doi.acm.org/10.1145/336512.336546.

LEITE, J. C. S. P. Gerenciando a Qualidade de Software com Base em Requistos. Disponível em: http://www.inf.puc-rio.br/~julio

MADACHY, R. J. System dynamics modeling of an inspection-based process. Proceedings of the 18th international conference on Software engineering, 1996.

MYLOPOULOS, J., CHUNG, L., AND YU, E. 1999. From object-oriented to goal-oriented requirements analysis. Commun. ACM42, 1 (Jan. 1999), 31-37. DOI=http://doi.acm.org/10.1145/291469.293165.

PAGLIUSOA, P.; TAMBASCIAA, C.; VILLAS-BOASA, A.; FREITASA, M.; LEVANTEZIA, M. GVR – Guia de Validação de Requisitos baseados nas técnicas PBR e ad-hoc resultante de um estudo de caso no CPqD1 a Fundação CPqD, Telecom & IT Solutions. Campinas, SP - Brasil.

PORTER, A. A.; VOTTA, L. G.; BASILI, V. R. Comparing detection methods for software requirements inspections: a replicated experiment. IEEE Trans. Software Engineering, v. 21, n. 6, p. 563-575, 1995.

ROBERT, B. G.; SLACK, T. V. Key Lessons in Achieving Widespread Inspection Use. IEEE Software, v. 11, n. 4 Los Alamitos, CA, USA: IEEE Computer Society Press, p. 46-57, 1994.

RUSSEL, G. W. Experience with Inspection in Ultra large-Scale Developments. IEEE Software, v. 8, n. 1, p. 25-31, 1991.

SAYÃO, M.; BREITMAN, K. K. Gerência de Requisitos. Faculdade de Informática da PUC-RS e DI/PUC-Rio. Departamento de Informática da/PUC-Rio. Rio de Janeiro, RJ.

SHULL, W. *ET AL*. What We Have Learned about Fighting Defects. International Software Metrics Symposium, Ottawa, Canada, 2002.

STERMAN, J. D. (2000) “Business dynamics: systems thinking and modeling for a complex world”. Boston, MA: Irwin McGraw-Hill, 2000. 982 p.

VENSIM (2010). Vensim from Ventana Systems, Inc. Disponível em: <http://www.vensim.com>. Acesso em: 3 de mar. de 2012.

WALIA, G.; CARVER, J. The Effect of the Number of Defects on Estimates Produced by Capture-Recapture Models. Proceedings of the 19th IEEE

International Symposium on Software Reliability Engineering - ISSRE'2008, Seattle, WA, USA, p. 305-306, 2008.