Pós-Graduação em Ciência da Computação

"Avaliação de Performabilidade de Riscos de Desenvolvimento em Projetos de *Software*"

Por

Alexsandro Marques de Melo

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE

2014

Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Alexsandro Marques de Melo

"Avaliação de Performabilidade de Riscos de Desenvolvimento em Projetos de *Software*"

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Prof. Dr. Eduardo Antônio Guimarães Tavares*

RECIFE
2014

Catalogação na fonte
Bibliotecária Joana D'Arc L. Salvador, CRB 4-572

Melo, Alexsandro Marques de.
Avaliação de performabilidade de riscos de desenvolvimento em projetos de software / Alexsandro Marques de Melo. – Recife: O Autor, 2014.
100 f.: fig., tab.

Orientador: Eduardo Antônio Guimarães Tavares.
Dissertação (Mestrado) - Universidade Federal de Pernambuco. CIN. Ciência da Computação, 2014.
Inclui referências.

1. Engenharia de software. 2. Avaliação de riscos. 3. Petri, Redes de. 4. Diagrama de bIocos. I.Tavares, Eduardo Antônio Guimarães (orientador). II. Título.

005.1 (22. ed.) MEI 2014-89

Dissertação de Mestrado apresentada por **Alexsandro Marques de Melo** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Avaliação de Performabilidade de Riscos de Desenvolvimento em Projetos de Software**" orientada pelo **Prof. Eduardo Antônio Guimarães Tavares** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Paulo Romero Martins Maciel
Centro de Informática / UFPE

_____

Prof. Gabriel Alves de Albuquerque Junior
Departamento de Estatística e Informática / UFRPE

_____

Prof. Eduardo Antônio Guimarães Tavares
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 27 de fevereiro de 2014.

_____

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

A Deus, o que seria de mim sem a fé que eu tenho nele.

Agradecimentos

A Deus, por ter me iluminado durante esses dois anos de curso e por me dar forças para lutar em busca de meus objetivos.

Ao professor Eduardo Tavares, pelas valiosas orientações e pela motivação para a superação das dificuldades e limitações.

Aos Professores Paulo Maciel e Gabriel Alves, por terem aceitado o convite para compor esta banca.

Aos demais professores do MoDCS, pelos ensinamentos transmitidos.

À minha esposa Wedillayne Marques, cujo apoio e compreensão tornaram possível a realização deste trabalho.

A Marcelo Marinho, gerente de projetos do HPCIn e Sismica, amigo que me deu grande apoio no desenvolvimento da minha pesquisa, agradeço pelo suporte e pelo fornecimento dos dados necessários para a realização prática deste trabalho.

Aos amigos Rosiberto, Anderson, Lubnnia, Danilo e Matheus pelos momentos de estudo, descontração e por todo o apoio e ajuda.

Agradeço ainda a todos aqueles que de alguma forma contribuiram para a realização deste trabalho, incentivando e apoiando.

"Cada sonho que você deixa pra trás, é um pedaço do seu futuro que deixa de existir".

—STEVE JOBS

Resumo

Falhas em projetos é um fator de destaque na abordagem feita pela comunidade de engenharia de *software* e muito tem sido feito em prol do sucesso desses projetos, porém, os riscos sempre existirão. O aumento das taxas de sucesso em projetos de *software* representa um desafio significativo para essa indústria, em que alguns riscos (por exemplo, atrasos no cronograma, aumento de custos) podem levar os projetos ao fracasso. Nesse contexto, a área de gerência de riscos tem uma importância significativa. No entanto, a falta de um processo de gerenciamento de riscos, aliada a estimativas deficientes de custo e de tempo, são algumas das principais causas das falhas dos projetos de desenvolvimento de *software*.

O gerenciamento de riscos contribui positivamente para a redução e controle dos riscos do projeto de *software*, através de sua identificação e quantificação. Foram propostas várias técnicas para avaliar os efeitos de tais problemas indesejáveis, mas estimativas de probabilidade são geralmente negligenciadas, e isso afeta uma avaliação adequada dos riscos. Por isso, o impacto de riscos no desempenho de um projeto de *software* é um aspecto importante que não deve ser desprezado.

Este trabalho propõe uma metodologia e modelos de dependabilidade e performabilidade para avaliação probabilística de riscos de desenvolvimento em projetos de *software*. Nesta metodologia, a avaliação de riscos é realizada utilizando diagramas de blocos de confiabilidade e redes de Petri estocásticas. Dois estudos de caso demonstram a viabilidade da técnica proposta. Com a aplicação da metodologia e dos modelos propostos, é possível verificar o impacto e avaliar a performabilidade dos riscos de desenvolvimento em projetos de *software*. Além disso, essa metodologia possibilitará a avaliação de outros riscos de desenvolvimento, bem como a avaliação de performabilidade em outras etapas do processo de desenvolvimento de *software*. Isso tudo pode ser utilizado pelos gerentes de projetos de *software* para avaliar o impacto dos riscos em diferentes projetos.

Palavras-chave: Avaliação de Dependabilidade, Avaliação de Desempenho, Avaliação de Performabilidade, Avaliação de Risco, Redes de Petri Estocástica, Diagramas de Bloco de Confiabilidade.

Abstract

Failures in software projects are a prominent factor in the approach taken by the software engineering community and much has been done for the success of such projects; however, there will always be risks. Increasing rates of success in software projects represent a significant challenge for the industry in which some risks (e.g., schedule delays, increased costs) may lead to failure in projects. In this context, the area of risk management has a significant importance. However, the lack of a process in risk management combined with deficient estimates of cost and time are some of the major causes of failures in software development projects.

The risk management contributes positively to the reduction and control of the risks in the software project through its identification and quantification. Various techniques were proposed to evaluate the effects of such undesirable problems, but probability estimates are generally neglected, thus affecting the proper assessment of the risks. Therefore, the impact of risk on the performance of a software project is an important aspect that should not be overlooked.

This paper proposes a methodology and models of dependability and performability for probabilistic assessment of development risks in software projects. In this methodology, the risk assessment is performed by using reliability block diagrams and stochastic Petri nets. Two case studies demonstrate the feasibility of the proposed technique. With the application of the methodology and proposed models, it is possible to verify the impact and assess the performability of development risks in software projects. Furthermore, this methodology allows the assessment of other development risks, as well as the performability evaluation in other stages of the software development process. All this can be used by software project managers to assess the impact of risks on different projects.

Keywords: Dependability Evaluation, Performance Evaluation, Performability Evaluation, Risk Assessment, Reliability Block Diagrams, Stochastic Petri Nets.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

BNCC *Bayesian Networks with Causality Constraints*

CMMI *Capability Maturity Model Integration.*

CTMC *Continuous Time Markov Chains*

DE *Developer*

FT *Fault tree*

MTTR *Mean Time to Repair*

JE *Junior Develope*

MTActivate *Mean Time to Activate*

MTPB *Mean Time Between Preventive Maintenances*

MTTF *Mean Time to Fail*

MTTR *Mean Time to Repair*

PN *Petri Nets*

PM *Preventive Maintenance*

PMI *Project Management Institute*

PMBOK *Project Management Body of Knowledge*

RBD *Reliability Block Diagram*

RG *Reliability Graph*

SFT *Software Fault Tree*

SRAEM *Software Risk Assessment and Estimation Model*

SRAEP *Software Risk Assessment and Evaluation Process*

SPN *Stochastic Petri Nets*

UML *Unified Model Language*

SE *Senior Develop*

SEI *Software Engineering Institute*

TR *Trainee*

XP *Extreme Programming*

Sumário

1 Introdução

A mente que se abre a uma nova ideia jamais voltará
ao seu tamanho original.
—ALBERT EINSTEIN

O mercado global de *software* cresce a cada ano, e espera-se um aumento no valor de US$ 396,7 bilhões em 2016 (um aumento de 35,4% desde 2011) (Guide 2012), o que exige decisões rápidas e melhoria constante dos processos de desenvolvimento. Com a intensificação da concorrência, as empresas de desenvolvimento de *software* têm se esforçado para produzir *software* com menor *time-to-market* para ampliar sua participação neste negócio bilionário. No entanto, ainda são comuns em empresas de desenvolvimento de *software*, problemas relacionados com a entrega do produto/serviço de *software*, em que orçamentos são extrapolados, levando à consequente falha dos projetos de *software*. A comunidade da engenharia de *software* tem alertado as empresas do ramo para alguns fatores que têm efetivamente ameaçado o sucesso dos projetos de *software*, tais como: requisitos e equipe.

Gerenciar projetos de *software* de forma bem sucedida pode garantir que determinadas empresas conquistem maior participação de mercado, particularmente considerando a possibilidade de serem impostas a estas empresas variáveis como competição intensiva, escassez de recursos, agilidade, menores custos e prazos.

Projetos de desenvolvimento de *software* possuem incertezas (riscos), dos quais o controle é um fator determinante para o sucesso ou fracasso de um projeto. Estudos recentes (Report 2010) indicam que apenas 32% dos projetos de *software* são entregues no prazo e cronograma estipulado. Do restante, 44% sofrem de atrasos com custos elevados e problemas de especificação de requisitos, 24% são cancelados.

Uma notória dificuldade para a comunidade científica de *software* é a entrega de produtos de *software* no prazo previsto e dentro do orçamento. A Causa principal para essas falhas é o fato de que todos os projetos estão sujeitos a riscos, e saber como tratá-los é um fator crucial para o seu sucesso.

Projetos de *software* falham devido a fatores de riscos (por exemplo, atrasos no cronograma e aumento dos custos), os quais podem causar grandes perdas em relação ao tempo, dinheiro e credibilidade no mercado. Assim, a indústria de *software* tem adotado metodologias, técnicas e ferramentas para atingir os objetivos do projeto no prazo e dentro do orçamento planejado. Muitos estudos enfatizam o gerenciamento de riscos como a principal causa do sucesso ou fracasso de projetos de *software*. Neste contexto, o gerenciamento de riscos contribui positivamente para o sucesso do projeto, mas esse gerenciamento ainda enfrenta obstáculos para serem inseridos nas metodologias das empresas de desenvolvimento de *software* (Bakker et al. 2009).

Este é um cenário normalmente encontrado no mercado de desenvolvimento de *software* e requer grande esforço por parte das empresas desse segmento. A partir deste cenário, surgiu a necessidade de as empresas de *software* desevolverem metodologias e técnicas para que seus projetos sejam mais previsíveis e que sejam entregues dentro dos prazos e orçamentos estabelecidos. Em sequência, surge a necessidade de adotar o gerenciamento de riscos no desenvolvimento de *software*, objetivando o sucesso dos projetos.

Riscos em um projeto de *software* ameaçam a sua viabilidade. Em outras palavras, sempre que os riscos se tornam reais, eles podem afetar consideravelmente a execução do projeto, ou mesmo levar a seu cancelamento. A categoria de risco proeminente (Guide 2013) são riscos de projeto (que contemplam os riscos de desenvolvimento), uma vez que pode afetar o cronograma do projeto ou os recursos necessários para o desenvolvimento de *software* (por exemplo, rotatividade de desenvolvedor).

A disciplina de gerenciamento de riscos é uma das mais importantes no processo de desenvolvimento de *software*, pois tem como finalidade identificar e mitigar os riscos de projetos, continuamente, através do ciclo de vida de um projeto. O gerenciamento de riscos permite aos gerentes de projetos de *software* alcançar seus objetivos no desenvolvimento de um projeto, contribuindo para um melhor tratamento das incertezas (riscos). Com o objetivo de controlar os riscos, a gerência de riscos sugere ações preventivas que promovam a mitigação, estimativas ou eliminação dos eventos de riscos identificados no projeto de *software*.

Na atualidade, com as limitadas ferramentas disponíveis no mercado de *software*, as

técnicas para o gerenciamento de riscos não conseguiram entrar no cotidiano da grande maioria das empresas de *software*. Dessa forma, tanto as ferramentas como as técnicas estão sendo aprimoradas e criadas constamente, buscando sempre o sucesso dos projetos de *software*.

Avaliação de riscos (Bakker et al. 2009) é geralmente realizada utilizando modelos qualitativos. No entanto, os modelos quantitativos podem ajudar os gerentes de projeto a estimar a ocorrência de riscos (por exemplo, estimativas de probabilidade) e, com isso, os gerentes de projetos podem realizar um melhor planejamento para evitar os riscos inerentes a projetos de *software*.

Avaliação de dependabilidade (Maciel et al. 2011) fornece várias técnicas baseadas em modelos formais (por exemplo, redes de Petri estocásticas (Balbo 2001) para estimar a ocorrência de falhas do sistema com base em estimativas de probabilidade. Em relação a projetos de *software*, também podem ser adotadas tais técnicas, fornecendo informações importantes para os gerentes sobre os riscos identificados no projeto.

Além disso, a avaliação de performabilidade (Araújo & et al. 2011) (ou seja, dependabilidade e desempenho) lida com o efeito de eventos de falha e atividades de reparação na degradação do desempenho do sistema. No contexto de projetos de *software*, a performabilidade fornece técnicas e modelos importantes para avaliar o impacto dos riscos na execução de processos de desenvolvimento.

1.1 Objetivos

O gerenciamento de risco é uma atividade fundamental na gestão de projetos, pois aumenta consideravelmente a chance de um projeto ser concluído com sucesso.

Projetos de *software* de sucesso (por exemplo, entregues no prazo e atendendo as restrições de custo) ainda são um desafio importante para a indústria. Neste contexto, os riscos merecem uma atenção especial, uma vez que podem levar ao fracasso de um projeto. Neste trabalho foram propostas várias técnicas para avaliar os efeitos de tais problemas indesejáveis, porém, como as estimativas de probabilidade são geralmente negligenciadas, isso afeta uma avaliação adequada dos riscos. Além disso, o impacto desses riscos no desempenho de um projeto de *software* é um aspecto importante que não deve ser desprezado.

Este trabalho propõe uma metodologia e modelos para avaliação de performabilidade de riscos de desenvolvimento em projetos de *software*. Esta metodologia apresenta uma série de etapas, que evoluem desde o modo de funcionamento/falha do projeto de *software*, definições de métricas e análise de dados até a geração do modelo e avali-

ação dos cenários. Os modelos levam em conta diagramas de blocos de confiabilidade (RBD) (Rausand & Høyland 2004, Kuo & Zuo 2003) e redes de Petri estocásticas (SPN) (Balbo 2001), considerando-se uma técnica de modelagem híbrida para calcular métricas de performabilidade. Este trabalho foca em riscos de desenvolvimento relacionados à rotatividade de desenvolvedor e à implementação de requisitos, riscos estes importantes e que afetam um projeto de *software* (Shahzad & S. 2010, Basit & Abdullah 2009, Izquierdo-cortazar & et al. 2010, Tracy & David 2008).

Com a apliação da metodogia e modelos propostos, é possível conceber vários cenários para avaliar o respectivo impacto sobre dependabilidade e performabilidade de riscos de desenvolvimento em projeto de *software*. Além disso, essa metodologia possibilitará a avaliação de diferentes riscos de projeto de *software*, por exemplo, os riscos relacionados à análise de requisitos.

Isso tudo pode ser utilizado pelos gerentes de projetos para estimar os riscos, adotando os modelos propostos e fornecendo informações importantes sobre os riscos identificados. Dessa forma, tal metodologia auxilia os gestores na tomada de decisões, bem como, na realizaão de um melhor planejamento para evitar e mitigar os riscos inerentes ao projeto de *software*.

1.1.1 Objetivos Específicos

O gerenciamento de riscos é uma atividade que merece uma atenção especial em projetos de *software* e, como consequência, vários modelos foram propostos para avaliar o impacto dos riscos durante a vida do projeto. Em geral, esses modelos nos tornam capazes de raciocinar sobre os impactos dos riscos, mas estimativas de probabilidade são geralmente negligenciadas, afetando uma avaliação adequada dos riscos. Neste contexto, o presente trabalho propõe uma metodologia para auxiliar os gerentes de projetos de *software* a avaliarem os riscos de desenvolvimento em projetos. Tal abordagem é baseada em modelos de dependabilidade e performabilidade (por exemplo, redes de Petri estocásticos e diagramas de bloco de confiabilidade) para avaliação probabilística dos riscos de desenvolvimento, relacionados à rotatividade de desenvolvedor e à implementação de requisitos. De forma mais específica, para se avaliar os riscos de desenvolvimento em projeto de *software*, o presente trabalho se propõe a:

- Propor uma metodologia que auxilie os gerentes de projetos de *software* a realizarem avaliação de performabilidade de riscos de desenvolvimento em projetos de *software*;

- Propor modelos de dependabilidade e performabilidade (SPN e RBD) para realizar avaliação probabilística dos riscos de desenvolvimento em projetos de *software*;
- Definir métricas para estimar o impacto dos riscos de desenvolvimento em projetos de *software*;
- Elaborar estudos de casos para avaliar e apresentar a adoção da metodologia e modelos propostos.

1.2 Estrutura do Documento

No Capítulo 2 é apresentada a contextualização dos trabalhos relacionados a esta dissertação.

O Capítulo 3 apresenta a fundamentação teórica em relação ao gerenciamento de riscos, avaliação de desempenho, avaliação de dependabilidade, avaliação de performabilidade, censura, diagramas de bloco de confiabilidade e redes de Petri.

No Capítuo 4 são apresentados a metodologia e os modelos propostos em SPN e RBD para estimar as métricas de dependabilidade e performabilidade. Posteriormente, um exemplo motivacional é descrito.

No Capítuo 5 são apresentados dois estudos de caso, baseados na metodologia e nos modelos propostos, para avaliação do impacto dos riscos de desenvolvimento em projetos de *software*. O estudo de caso 1 trata sobre a rotatividade de desenvolvedor como um risco potencial, já o estudo de caso 2 tem a implementação de requisitos como risco relacionado.

O Capítulo 6 apresenta as conclusões desta dissertação, assim como as principais contribuições e propostas para trabalhos futuros.

2

Trabalhos Relacionados

É preferível saber poucas coisas muito
bem a saber muitas coisas muito mal.
— CID CERDAL

Este capítulo tem por finalidade apresentar alguns trabalhos representativos para esta dissertação, destacando-se alguns trabalhos que realizam avaliação quantitativa de riscos em projetos de *software*, relacionados ao contexto dessa dissertação, e também alguns trabalhos na área de gerenciamento de riscos em projetos de *software* para avaliação de desempenho.

2.1 Avaliação Quantitativa de Riscos

Atualmente, desenvolver *software* com menor *time-to-market* é uma necessidade do mercado global. A adoção da disciplina de gerenciamento de riscos em projetos de *software* vem crescendo e é uma atividade de destaque na gestão de projetos de *software* e, como consequência, vários modelos foram propostos para avaliar o impacto dos riscos durante a vida do projeto. Esses modelos são capazes de prever os impactos dos riscos, mas estimativas de probabilidade são geralmente negligenciadas e, consequentemente, afetam uma avaliação adequada dos riscos.

Avaliação quantitativa (Bakker et al. 2009) de riscos em projetos de *software* não é uma prática comum. No entanto, a comunidade de engenharia de *software* necessita de técnicas de avaliação quantitativa, e algumas obras representativas foram desenvolvidos ao longo dos anos.

Gupta e Sadiq (Gupta & Sadiq 2008) propõem modelo de Avaliação de Risco e Estimativa (SRAEM) para avaliar os riscos em projetos de *software*, eles consideram alguns

fatores de risco, tal como: modificação de requisitos (por exemplo, adição, exclusão). Tal técnica adota pontos de função, erro de medição, erro de modelo e erro de suposição para quantificar ocorrência de risco. Este modelo avalia o risco incremental para cada fase do processo de desenvolvimento de *software*. O modelo é útil apenas se tivermos as informações sobre os pontos de função do projeto de *software*.

Wattanapokasin e Rivepiboon (Wattanapokasin & Rivepiboon 2009) propõem um modelo matemático para estimar os riscos relacionados às diferenças culturais em uma equipe de desenvolvimento de *software*, tais como: a língua falada, estilo de comunicação, método de desenvolvimento e fusos horários diferentes. A técnica assume dados históricos ou dados fornecidos pelo gerente do projeto e um processo de Poisson para estimar probabilidades de risco. O modelo é limitado aos riscos citados anteriormente, mas poderiam incluir outros fatores de riscos culturais, tais como comunicação entre *stakeholders*, dentre outros.

Falahah (Falahah 2011) descreve um método para quantificação de risco, com foco em três elementos de risco: (i) risco técnico, (ii) risco de custo, e (iii) risco de programação. O método fornece estimativas de probabilidade através de um questionário como dados de entrada. O método estima o risco em cada fase do projeto de *software* à medida que progride de fase para fase. Este método não leva em conta as questões da complexidade do *software*, que desempenham um papel importante na determinação do risco para os projetos de *software*. O método proposto também não leva em conta as questões relacionadas com os requisitos (por exemplo, alteração).

Em (Hu & et al. 2012), os autores propõem um modelo baseado em redes bayesianas com restrições de causalidade (BNCC) para análise de risco de projetos de desenvolvimento de *software*. O modelo leva em conta alguns fatores de risco relacionados (por exemplo, requisito, equipe e usuários do sistema). O modelo se concentra em encontrar a correlação entre fatores de risco e os resultados do projeto.

Em (Boness & et al. 2008), Boness et al. apresentam uma técnica para avaliar o risco do projeto de *software* e perdas relacionadas durante a fase de análise de requisitos. Essa técnica realiza a avaliação dos riscos por meio de métricas subjetivas de alto nível, coletadas durante a análise de requisitos e utiliza gráficos meta para estimativas de probabilidade.

Rousan et al. (Al-Rousan 2009) apresentam uma ferramenta para avaliar probabilisticamente os riscos em projetos de *software web*, através de redes bayesianas e, a partir de dados de projetos anteriores, a ferramenta concentra-se nos fatores de riscos (por exemplo, requisitos e *stakeholders*).

Em (qiu Liu et al. 2012) os autores propõem um método para a avaliação de risco de projeto de *software*. O método leva em conta alguns fatores de riscos (por exemplo, técnicos e custos). Esse método realiza avaliação dos riscos através de fórmulas fechadas e pesos são atribuídos aos fatores de riscos.

Em (Amrit & Mark 2004) os autores apresentam uma ferramenta para avaliação de risco durante o desenvolvimento de *software*, tendo em conta seis fatores de risco: (i) uso de uma metodologia inadequada, (ii) falta de envolvimento do cliente, (iii) falta de práticas formais de gerenciamento de projetos, (iv) dissimilaridade de projetos anteriores, (v) complexidade do projeto, e (vi) volatilidade de requisitos. O usuário associa pesos a cada risco especificado, de modo que a ferramenta fornece um valor de 10 (baixa probabilidade) a 100 (alta probabilidade) para estimar a ocorrência de risco.

Da mesma forma, Mustafa et al. (Mustafa & Jalil 2010) propõem uma ferramenta de avaliação de risco, concentrando-se sobre os riscos relacionados com os requisitos do produto e processo de desenvolvimento. A partir de um conjunto de pesquisas, um valor é obtido entre 1 (baixo) e 3 (alto) para estimar a probabilidade de risco.

Sadig et al. (Sadiq 2010) apresenta a abordagem SRAEP que leva em conta SFT para estimar e priorizar riscos em projetos de *software*. Essa abordagem identifica e analisa os riscos do projeto de desenvolvimento de *software*. A abordagem contempla alguns fatores de riscos (por exemplo, requisitos).

Tang e Wang (Tang & long Wang 2010) apresentam um modelo para avaliação de riscos em projetos de *software*, no qual alguns fatores de riscos são abordados (por exemplo, tecnologia utilizada, complexidade do *software* que está em desenvolvimento). O modelo propõe uma hipótese de que o risco do projeto de *software* é baseado na Teoria dos Conjuntos *Fuzzy*. Eles também adotam a lógica *fuzzy* para avaliar e calcular a probabilidade dos riscos e seus respectivos impactos.

Em (Foo & Muruganantham 2000) os autores propõem o Modelo (SRAM) para avaliação de riscos em projetos de desenvolvimento de *software*, que abrange nove elementos críticos (i) complexidade do *software*, (ii) pessoal envolvido no projeto, (iii) confiabilidade alvo, (iv) requisitos do produto, (v) método de estimativa, (vi) método de monitoramento, (vii) processo de desenvolvimento adotado, (viii) usabilidade do *software*, e (ix) ferramentas utilizadas para o desenvolvimento. Um questionário para avaliar a probabilidade de um risco é especificado. Um conjunto de perguntas é cuidadosamente escolhido para cada um desses elementos com três alternativas de respostas cada. As respostas estão dispostas em ordem crescente de risco. Este modelo considera o método de priorização como uma única etapa de avaliação de risco, mas não especifica como a

priorização seria feita.

Em (Solutions 2006) é apresentada uma ferramenta de gestão de riscos para ambientes de múltiplos projetos de desenvolvimento de *software*, que utiliza componentes de inteligência artificial e ontologia fundamentada na taxonomia de riscos do SEI. Tal ferremanta é aderente ao modelo CMMI garantindo, desta forma, a qualidade do processo de desenvolvimento. A ferramenta auxilia os gerentes de projeto na execução das atividades de identificação, monitoração e controle dos riscos.

Knob et al. (Knob & et al. 2006) propõem uma ferramenta que tem como objetivo auxiliar equipes de projetos nas tarefas relacionadas à gerência de riscos em projetos de *software*. A ferramenta foi projetada para estar em comformidade com os objetivos e práticas sugeridos pelo PMBOK e CMMI.

Ao longo dos últimos anos têm sido propostos alguns trabalhos científicos na área de gerenciamento de riscos em projetos de *software* para avaliação de desempenho. Entretanto, vários trabalhos representativos avaliam o desempenho dos projetos e processos de *software*.

Em (Zowghi & Nurmuliani 2002), os autores apresentam um modelo para avaliar o impacto da volatilidade dos requisitos no desempenho dos projetos de *software*. A volatilidade dos requisitos é caracterizada pelas diferenças ou divergências e os conflitos entre os usuários/*stakeholders* sobre os requisitos. Os autores definem que a volatilidade dos requisitos está associada negativamente com o cronograma, custo e desempenho do projeto de *software*. O modelo fornece estimativas de probabilidade usando análise de regressão e questionário como dados de entrada.

Em (Yan & Yu-feng 2011), os autores apresentam um modelo de previsão estatística para estimar o desempenho no processo de desenvolvimento de *software*, tendo em conta equações de forma fechada com base em análise de regressão. O modelo assume dados históricos para estimar algumas métricas (por exemplo, média do nível de habilidade da equipe e cobertura de teste).

Duarte et al. (Duarte & Raza 2012) propõem uma ferramenta para analisar quantitativamente o desempenho dos desenvolvedores de *software* com base em dados históricos e fórmulas fechadas. As métricas de interesse incluem a produtividade e o tamanho da equipe.

Chen et al. (Pei-Chi & C. 2012) propõem um modelo de regressão linear para avaliar a relação entre as características da equipe do projeto de *software* e desempenho da equipe. O modelo define um conjunto de hipóteses (por exemplo, existe relação negativa entre a diversidade da equipe e desempenho do projeto, existe relação positiva entre a

flexibilidade da equipe e o desempenho do projeto) para avaliar tais questões.

Marinho et al. (Marinho & et al. 2010) propõem um método para avaliação de desempenho de processos de testes de *software*. De Diagramas UML, um mapeamento é definido, a fim de gerar modelos de redes de Petri estocásticas. Utilizando o modelo, é possível avaliar o impacto das mudanças do processo de teste de *software* e simulações são adotados para obtenção de estimativas. Além disso, o método possibilita avaliar diferentes alternativas de implementações, bem como a verificação de melhor composição de recursos humanos para as atividades do processo de teste de *software*.

Em (A. Schmietendorf & Rautenstrauch 2000) os autores propõem um modelo para avaliar o desempenho dos processos de *softwares*. O modelo se baseia no CMMI do SEI e é baseado em um catálogo de questionários para avaliar o desempenho dos processos de *softwares*. Dessa forma, com esse modelo é possível avaliar processos de *softwares* com o termo de desempenho. O modelo proposto auxilia no estabelecimento de processos para prover um bom desempenho das atividades desse processo.

Diferente dos trabalhos anteriores, essa dissertação apresenta uma abordagem baseada em modelos de dependabilidade e desempenho (performabilidade), para estimar os riscos de desenvolvimento em relação à rotatividade de desenvolvedor e implementação de requisitos. Além disso, a abordagem proposta também considera técnicas de tolerância a falhas para mitigar o impacto dos riscos.

2.2 Considerações Finais

Este capítulo apresentou alguns trabalhos científicos representativos para esta dissertação. Em tais trabalhos, é apresentada grande variedade de métodos e modelos que são utilizados para avaliação quantitativa de riscos em projetos de *software*, bem como para avaliação de desempenho em projetos de *software*.

3
Fundamentação Teórica

Se não puder vencer pelo
talento, vença pelo esforço.
—MAX BEERBOHM

Este capítulo apresenta os conceitos sobre gerencimento de riscos, incluindo as categorias de riscos e as atividades do gerenciamento de riscos. Além disso, são abordados conceitos sobre avaliação de desempenho, avaliação de dependabilidade, avaliação de performabilidade e censura. Em seguida são apresentados os diagramas de bloco de confiabilidade, tais como os arranjos adotados. Posteriormente, é descrita uma introdução sobre redes de Petri, assim como definições, conceitos básicos e propriedades, as quais podem ser divididas em duas categorias: propriedades comportamentais e propriedades estruturais. Por fim, são apresentadas as redes de Petri estocásticas (SPNs), que é a extenção de redes de Petri proeminentemente adotada neste trabalho.

3.1 Gerenciamento de Riscos

O desenvolvimento de *software* é uma atividade complexa, por causa de inúmeros fatores de riscos, como atrasos no cronograma, aumento dos custos, dentre outros. No entanto, esta complexidade faz com que grande parte dos projetos de *software* extrapolem o prazo e orçamento previstos. Um gerenciamento eficaz tem fundamental importância para o sucesso de projetos de *software*.

Um dos objetivos da gerência de projetos é prever os riscos que podem afetar o bom andamento do projeto e definir ações a serem tomadas para conter sua ocorrência ou, quando não for possível evitar a ocorrência, reduzir seus impactos.

Podemos pensar no risco como alguma circunstância adversa. Eles podem ameaçar o projeto, o *software* que está sendo desenvolvido ou a organização (Somerville 2011). Gerenciá-los, portanto, é uma questão essencial para o sucesso dos projetos de *software*.

As empresas de desenvolvimento de *software* lidam com riscos e necessitam gerenciá-los constantemente, como forma de antecipar e minimizar o efeito de eventos indesejáveis que possam impactar negativamente nos projetos de *software* e, consequentemente, levar à falha do projeto.

O gerenciamento de riscos é uma técnica empregada na engenharia de *software*, o qual representa um grande instrumento para a gerência desses projetos. O gerenciamento de riscos, basicamente aumenta a probabilidade e o impacto de eventos positivos e diminui a probabilidade e o impacto dos eventos negativos do projeto de *software* (Guide 2013). Assim, seu grande objetivo é ajudar os gerentes de projeto de *software* a entender e gerenciar incertezas durante o desenvolvimento do *software*. O seu uso, reune ações como identificação de incertezas (riscos), cálculo das probabilidade e dos impactos, desenvolvimento de respostas para eliminar, reduzir ou lidar com riscos. O controle sobre tais incertezas é um forte fator de determinação do sucesso ou fracasso. A gerência de riscos, portanto, é crucial para um bom gerenciamento de projeto de *software* (Pressman 2006).

São vários os tipos de riscos que podem afetar a execução de um projeto de *software*, tais como, riscos de rotatividade de membros da equipe, em que pessoas fundamentais do projeto deixam-no antes do término; riscos relacionados a mudanças de requisitos, podendo afetar o prazo, cronograma e custos do projeto. Além destes, riscos de mudanças de tecnologias, em que a tecnologia básica do sistema foi superada por uma nova tecnologia. Assim, o gerenciamento de riscos é uma aréa de fundamental importância na gestão de projetos de *software*, porque aumenta consideravelmente a chance de um projeto obter sucesso.

Os riscos em projetos de *software* são divididos em 3 categorias (Pressman 2006):

- Riscos de projeto: são os riscos que afetam o cronograma ou os recursos dos projetos. Se os riscos se tornarem reais, o tempo e o custo do projeto tendem a aumentar. Os fatores relacionados a estes riscos são: pessoal, recursos, clientes e requisitos. Dessa forma, podem atrasar o cronograma e aumentar os custos;

- Riscos de produto: são os riscos que afetam a qualidade ou o desempenho do *software* que está em desenvolvimento. Se os riscos se tornarem reais, a implementação do projeto pode se tornar inviável. Tais riscos, envolvem problemas

de linguagem de programação utilizadas, interface, *design*, manutenção, dentre outros, consequentemente, podem ameaçar a qualidade do projeto;

- Riscos de negócio: são os riscos que afetam a organização que está desenvolvendo ou adquirindo o *software*. Caso os ricos se tornem reais, o projeto pode tornar-se inviavél e até cancelado. Os riscos de negócio são: substituição do gerente do projeto, desenvolvimento de um *software* que não se adequa ao mercado e desenvolvimento de um *software* sem demanda.

Gerenciamento dos riscos de um projeto inclui um processo que trata do planejamento, identificação, análise, resposta, monitoramento e controle dos riscos em um projeto (Guide 2013), prevendo, assim, os riscos que podem afetar o sucesso do projeto de *software* e, consequentemente, permitindo que se tomem as medidas necessárias para evitar os riscos. O processo de gerenciamento de riscos é um processo iterativo, que continua ao longo do projeto (Somerville 2011).

O processo de gerenciamento de riscos envolve várias atividades (Somerville 2011, Guide 2013):

- Identificação de riscos: são identificados os possíveis riscos de projeto, produto/-serviço e negócios;
- Análise de riscos: são avaliadas as possibilidades e as consequências da ocorrência desses riscos.
- Planejamento de riscos: são traçados planos para enfrentar os riscos, seja evitando-os, seja minimizando seus efeitos sobre o projeto;
- Monitoramento e controle de riscos: o risco é constantemente avaliado e os planos para a sua diminuição são revisados, à medida que mais informações sobre eles se tornem disponíveis.

Boa parte dos riscos de projetos de *software* pode ser minimizada com a definição de um processo de gerenciamento de riscos que esteja de acordo com a realidade da empresa de *software* onde se deseja implementar o gerenciamento de riscos.

Desta forma, os métodos e modelos, disponíveis na literatura de Engenharia de *Software*, referentes ao processo de gerenciamento de riscos utilizam estas atividades.

O gerenciamento de riscos em projetos de *software* tornou-se uma atividade "chave"no desenvolvimento de *software*, pois introduz respostas adequadas com rapidez, para influenciar positivamente no resultado do projeto.

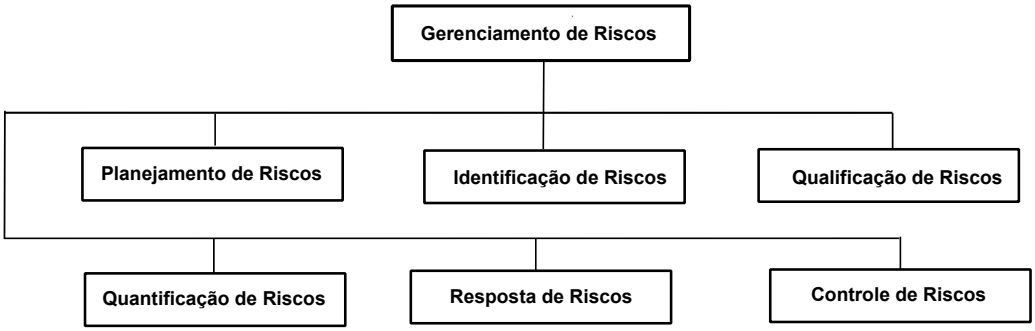

Figura 3.1: Atividades do Gerenciamento de Riscos Adaptados PMI (Guide 2013).

O propósito do gerenciamento de riscos é identificar riscos assim que possível, estabelecer estratégia no desenvolvimento do *software* para mitigar e/ou eliminar estes riscos, criar e executar um processo de gerenciamento como uma parte global do processo de desenvolvimento de *software*.

3.2 Avaliação de Desempenho

O desempenho é geralmente definido como o grau em que o sistema realiza suas funções designadas dentro de determinados limites (por exemplo, a velocidade) (Trivedi & et al. 2009). A avaliação de desempenho (Hermanns et al. 2002) tem sido de grande importância nas pesquisas de desenvolvimento e otimização de sistemas, ou seja, uma atividade essencial para promover melhorias na qualidade dos mesmos.

A avaliação de desempenho é utilizada para a avaliação quantitativa de sistemas. A avaliação se relaciona à descrição, à analise e à otimização do comportamento dinâmico e dependente do tempo dos sistemas (Hermanns et al. 2002). Dessa forma, a avaliação de desempenho nos possibilita lidar com problemas frequentemente encontrados em sistemas computacionais (por exemplo, identificação de gargalos, carga de trabalho e comparação de sistemas).

A avaliação de desempenho geralmente contempla algumas métricas, tais como:

- Vazão: é o número de operações que são executadas em um determinado período de tempo;
- Utilização: representa a porcentagem de tempo que o sistema está ocupado realizando uma atividade (ação). O recurso com a maior utilização pode ser considerado o gargalo do sistema;

- Tempo de resposta: compreende ao tempo entre a chegada de uma requisição e a resposta do sistema.

A avaliação de desempenho de sistemas computacionais consiste de um conjunto de critérios e técnicas classificadas como as baseadas em medição e modelagem. As técnicas baseadas em modelagem podem ser classificadas como técnica analítica e simulação (Lilja 2000).

A medição de desempenho consiste essencialmente na monitoração do sistema enquanto está sob a ação de uma carga de trabalho (Jain 1991). Para adquirir resultados representativos, a carga de trabalho deve ser cuidadosamente selecionada pelos projetistas e utilizada nos estudos de desempenho, podendo ser real ou sintética (Jain 1991, Lilja 2000).

Verifica-se, porém, que a carga de trabalho real não é adequada para utilização devido à impossibilidade de repetição. Isso acontece quando o tamanho da carga não é considerável e também quando esses dados receberam muitas perturbações ou, até mesmo, por questões de acessibilidade. Por esses motivos, uma carga sintética, cujas características são semelhantes às da carga de trabalho real, pode ser aplicada repetidamente de uma maneira controlada, desenvolvida e usada para estudos.

A essencial razão para a utilização de uma carga de trabalho sintética é que ela é uma representação ou modelo da carga de trabalho real. A carga de trabalho pode ser simplesmente modificada sem afetar a operação e pode ser facilmente portada para sistemas diferentes, graças ao seu pequeno tamanho e ela pode ter embutida capacidades de medição (Jain 1991).

A modelagem analítica utiliza um conjunto de equações e funções matemáticas para descrever o comportamento de um sistema. Durante a construção dos modelos, deve-se levar em consideração a complexidade e praticidade dos sistemas.

Os modelos analíticos permitem uma análise em relação aos efeitos causados pelos parâmetros definidos nas equações sobre a aplicação. Ademais, também se pode estabelecer possíveis relacionamentos entre cada um dos parâmetros considerados. Para validar os resultados alcançados por meio dos modelos elaborados, a modelagem analítica pode compará-los aos valores reais medidos em testes experimentais (Jain 1991).

A simulação é utilizada tanto em avaliação de desempenho, quanto na validação de modelos analíticos. Ao contrário das medições analíticas, as simulações baseiam-se em modelos abstratos do sistema, por isso não exigem que o sistema esteja totalmente implantado para que sejam aplicadas. Desta forma, os modelos utilizados durante a simulação são desenvolvidos através da abstração de características essenciais do sistema,

sendo que a complexidade e o grau de abstração dele podem mudar de um sistema para outro. Durante a simulação, controlam-se, com maior eficiência, os valores assumidos por parâmetros do sistema. Assim sendo, fica mais fácil obter informações relevantes para a avaliação de desempenho.

3.2.1 Modelos

Existem diversos tipos de modelos para a avaliação de desempenho, tais como: Redes de Filas (Cassandras & Lafortune 2008), Cadeias de Markov (Trivedi 2001, Murata 1989) e Redes de Petri Estocásticas (Balbo 2001), que são modelos baseados em estados, ou seja, podem ser também definidos como não combinatórias. Esses modelos têm sido utilizados para estimativa de métricas de desempenho.

3.2.1.1 Redes de Filas

As Redes de Filas (Cassandras & Lafortune 2008) são uma das mais populares técnicas de modelagem utilizadas para a análise de desempenho de sistemas computacionais. As redes de fila descrevem processo de chegada de cliente/produto a um sistema de atendimento (por exemplo, beneficiamento ou produção) para receber um ou mais serviços, executados por determinada quantidade de servidores. Neste contexto, as formações de filas ocorrem em razão de a procura pelo serviço prestado ser maior do que a capacidade do sistema de atender a esta procura. A Figura 3.2 apresenta um sistema de filas.

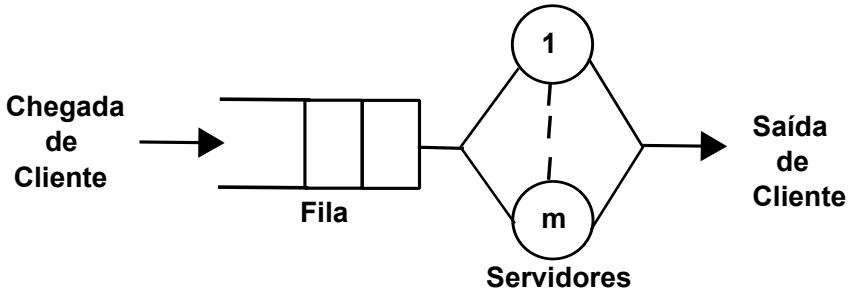

Figura 3.2: Sistema de Filas Adaptado de (Bolch et al. 2006).

A notação a seguir, conhecida como notação de Kendall (Bolch et al. 2006), é muito utilizada para decrever sistemas de filas:

$$A/B/m/K \tag{3.1}$$

na qual, A indica a distribuição da chegada de cliente, B denota a distribuição do

tempo de serviço, e m indica o número de servidores (m 1) e K indica o tamanho máximo da fila (capacidade). Os seguintes símbolos são normalmente utilizados para A e B:

- M: Distribuição exponencial (Memoryless);
- D: Distribuição determinística, ou seja, o tempo entre o serviço;
- E_k: Distribuição Erlang com k-fases;
- H_k: Distribuição Hiperexponencial com k-fases

Por exemplo, $M/M/1/4$ indica que o tempo entre chegadas e o tempo de serviço é regido por distribuições exponenciais, há apenas 1 servidor e a capacidade da fila é de 4 clientes.

Para avaliar o comportamento de sistemas de filas, associam-se medidas de desempenho tais como: tempo médio de espera dos clientes na fila, tempo médio de chegada de clientes, probabilidade de encontrar o sistema lotado, capacidade do sistema, entre outras. Dessa maneira, a teoria das filas tenta, por meio de análises matemáticas detalhadas, encontrar um ponto de equilíbrio que satisfaça o cliente ou linha de produção e seja viável para o provedor do serviço.

3.2.1.2 Cadeias de Markov

As Cadeias de Markov (Trivedi 2001, Murata 1989) é um formalismo matemático utilizado para a modelagem de sistemas. Assim, o formalismo permitem descrever o funcionamento de um sistema utilizando um conjunto de estados e transições entre esses estados. As transições entre os estados são modeladas por um processo estocástico de tempo contínuo ou discreto definido por distribuições exponenciais.

Um modelo descrito pelo formalismo de Cadeias de Markov pode ser interpretado como uma máquina de estados, onde os nodos da cadeia representam os estados e os arcos representam as transições.

Um modelo descrito pelo formalismo de Cadeias de Markov (Bolch et al. 2006) pode ser classifacado de acordo com a sua escala e tempo:

- Cadeias de Markov à escala de Tempo Contínua (CTMC - Continuos Time Markov Chains);
- Cadeias de Markov à escala de Tempo Discreta (DTMC - Discrete Time Markov Chains);

Os modelos em CTMC diferem dos modelos em DTMC basicamente por suas transições entre os estados poderem ocorrer em qualquer instante de tempo e não em pontos discretos de tempo.

A seguir, apresenta-se as propriedades para a construção de um modelo descrito pelas Cadeias de Markov (Stewart 1994).

Os estados do modelo são discretos e enumeráveis. Dessa forma, o formalismo de Cadeias de Markov permite cadeias de infinitos estados. A escala de tempo para a transição entre os estados do modelo pode ser de forma contínua (CTMC) ou discreta (DTMC).

A transição entre os estados do modelo depende exclusivamente do estado atual do modelo, sem importar quais foram os estados prévios ou serão os estados futuros do modelo. A taxa (CTMC) ou probabilidade (DTMC) de transição de estados do modelo dá-se obedecendo a uma lei exponencial ou geométrica, respectivamente.

A representação gráfica de um modelo em Cadeias de Markov é feita por autômatos, à qual é associada para cada estado do autômato um estado do modelo e para cada transição uma taxa (CTMC) ou uma probabilidade (DTMC). Um modelo em Cadeias de Markov é representado, matematicamente, por uma matriz de transição de estados. A probabilidade de cada estado em regime estacionário (solução de um modelo em Cadeias de Markov) é a solução do sistema da equação linear 3.3:

$$Q = \begin{pmatrix} q_{ii} & q_{ij} \\ q_{ji} & q_{jj} \end{pmatrix}, \tag{3.2}$$

$$\pi Q = 0 \tag{3.3}$$

na qual, Q é a matriz de transição de estados e π (vetor de probabilidade) é o autovetor correspondente ao autovalor unitário da matriz de transição. É importante ressaltar que a soma dos elementos do vetor de probabilidade π deve ser igual a 1, ou seja, $||\pi|| = 1$ (Araújo 2009).

Para os modelos em CTMC, a matriz de transição de estados Q é denominada de gerador infinitesimal, no qual, cada elemento não diagonal da linha *i* e coluna *j* da matriz representa a taxa de transição do estado i para o estado *j* do modelo. Os elementos diagonais de *Q* representam o ajuste necessário para que a soma dos elementos de cada linha seja igual a zero (Araújo 2009) .

Para os modelos em DTMC, a matriz de transição de estados P é denominada de matriz estocástica, na qual cada elemento não diagonal representa a probabilidade de

transição entre os estados do modelo (Araújo 2009). Os elementos diagonais de P representam o ajuste necessário para que a soma dos elementos de cada linha seja igual a um.

A Figura 3.3 apresenta um modelo em CTMC com dois estados e duas transições. Cada transição entre um estado e outro possui associada uma taxa de ocorrência.

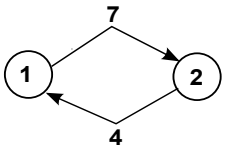

Figura 3.3: Modelo em CTMC com dois estados e duas transições.

3.2.1.3 Redes de Petri Estocásticas

Uma Rede de Petri Estocásticas (SPNs) (Balbo 2001) é uma abstração de um sistema. Ela é um modelo formal do fluxo de dados e controle do sistema modelado em questão. As propriedades, conceitos e técnicas para modelagem de uma Rede de Petri foram desenvolvidas utilizando métodos simples para descrição e análise do fluxo de sistema.

O formalismo de redes de Petri é utilizado principalmente em sistemas que possam apresentar atividades assíncronas, concorrentes e não-determinísticas. As SPNs permitem a modelagem e análise probabilística de sistemas. As transições em SPNs podem ser imediatas ou temporizadas. As transições temporizadas possuem um atraso exponencialmente distribuídos. As SPNs têm sido amplamente utilizadas para avaliações de desempenho de sistema. As Redes de Petri Estocásticas foram descritas na seção 3.9.

3.3 Avaliação de Dependabilidade

A avaliação de dependabilidade é uma atividade essencial que tem como objetivo fornecer meios para que seja possível promover melhoria da qualidade dos serviços prestados. Com o aumento dos serviços oferecidos pela internet, a dependabilidade tornou-se um atributo de grande importância no desenvolvimento de *software* e *hardware*, na implantação e operação dos serviços oferecidos (Maciel et al. 2011).

Dependabilidade é a propriedade que define a capacidade de o sistema prestar um serviço que pode justificadamente ser confiável (Kuo & Zuo 2003, Avizienis 2001). Dessa forma, dependabilidade representa a capacidade de um sistema em oferecer um serviço de forma confiável (Maciel et al. 2011). Um conceito importante é a falha do sistema,

o que acontece quando o sistema para de fornecer as respectivas funcionalidades. Uma falha refere-se a um defeito de um componente do sistema (ou subsistema), o que pode causar outros defeitos ou falhas no sistema.

Em geral, os conceitos de dependabilidade são: atributos, meios e ameaças (Avizienis 2001). A Figura 3.4 apresenta a árvore de dependabilidade, na qual:

- Os atributos: possibilitam a obtenção de medidas quantitativas, em que muitas são cruciais para a análise dos serviços prestados;
- Os meios: são os meios pelos quais a dependabilidade é atingida;
- As ameaças: compreendem as falhas, erros e defeitos. A falha do sistema é o evento que ocorre quando a entrega do serviço não acontece de forma desejada.

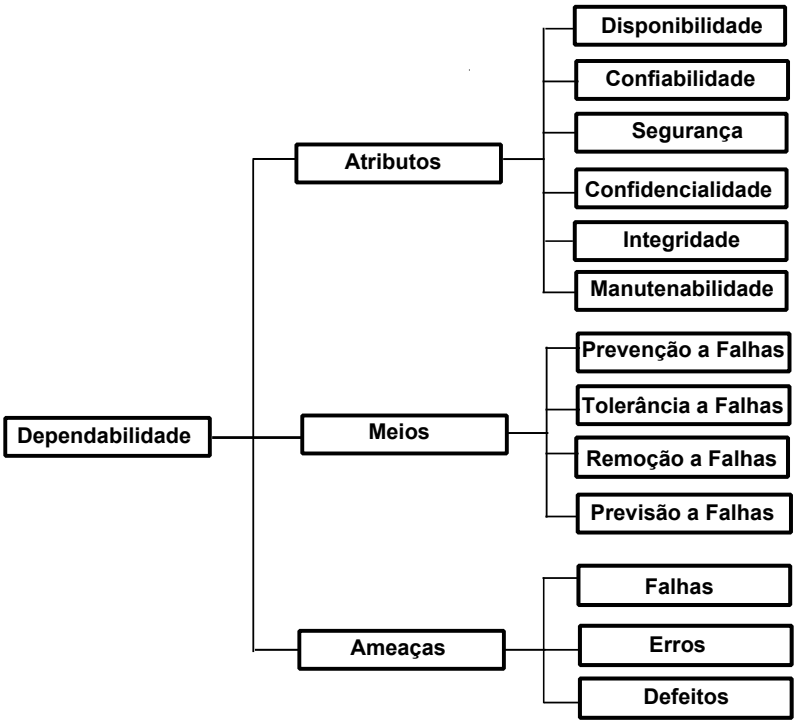

Figura 3.4: Árvore de dependabilidade Adaptado de Avizienis (2001).

A dependabilidade usualmente contempla as seguintes métricas/atributos: confiabilidade, disponibilidade, manutenabilidade, segurança, confidencialidade e integridade. Neste trabalho, os atributos de interesse são:

- Confiabilidade: probabilidade de um sistema fornecer suas funções pré-definidas, sem falhas, por um período de tempo específico (Maciel et al. 2011): Matematicamente, é apresentada pela Equação 3.4.

$$R(t) = P\{T \geq t\} \tag{3.4}$$

, em que T é a variável aleatória que representa o tempo de falha do sistema (ou um único componente). De fato, $R(t) = 1 - F(t)$, tal que $F(t) = P\{T < t\}$ é a função de distribuição cumulativa, declarando que uma falha ocorre antes do tempo t;

- Disponibilidade: probabilidade de um sistema estar em uma condição de funcionamento. Ele considera a alternância de estados operacionais e não operacionais (Maciel et al. 2011). A Disponibilidade estacionária (A) pode ser representada palas seguintes equações 3.5 e 3.6:

$$A = \frac{uptime}{uptime + downtime} \tag{3.5}$$

em que o tempo de atividade (*uptime*) é o período de tempo em que o sistema está operacional e o tempo de inatividade (*downtime*) corresponde ao período de tempo em que o sistema não está operacional.

Ou ainda:

$$A = \frac{MTTF}{MTTF + MTTR} \tag{3.6}$$

no qual, o MTTF é o tempo médio de falha e MTTR é o tempo médio de reparo, de tal modo que

$$MTTF = \int_0^\infty R(t)dt \tag{3.7}$$

$$MTTR = \int_0^\infty (1 - M(t))dt \tag{3.8}$$

$M(t)$ é a função de distribuição cumulativa que representa a probabilidade de que um reparo ocorrerá dentro do tempo t. $R(t)$ é a função de confiabilidade como anteriormente apresentada.

Durante a execução do sistema, remoção de falhas poderão ser realizadas através de políticas de manutenção, tais como manutenção corretiva e preditiva. Manutenção corretiva restaura um componente/sistema após uma falha, enquanto a manutenção preditiva tenta manter o sistema em um estado operacional, evitando que falhas ocorram.

3.3.1 Técnicas Tolerantes a Falhas

Sempre que um sistema fornece as suas funcionalidades na presença de falhas, o sistema é considerado tolerante a falhas.

A redundância é uma conjunto de técnicas importante para a implementação de sistemas tolerantes a falhas com o objetivo de melhorar a disponibilidade e confiabilidade (Maciel et al. 2011). Em geral, tal técnica consiste em adicionar componentes externos para o sistema, de tal modo que, se um componente falha, o componente redundante assumirá seu lugar. Redundância dinâmica é uma técnica representativa que leva em conta os componentes de reposição para substituir o componente principal sempre que é inoperante. *Hot* e *cold standby* são abordagens representativas (Eric Bauer & Eustace 2011).

Considerando *cold standby*, um componente de *backup* só é ativado quando o componente primário falhar. Por outro lado, o componente *Hot standby* fica em execução simultaneamente com o componente primário. Sempre que o último falhar, o *backup* imediatamente o substituirá.

3.3.2 Técnicas de Modelagem

Modelos para estimar atributos de dependabilidades são geralmente classificados como modelos combinatórios ou modelos baseados em estado (Maciel et al. 2011). Modelos combinatórios consideram as condições que tornam o sistema operacional ou com falhas, a respeito da relação estrutural entre seus componentes. No entanto, esses modelos têm limitações para representar interações complexas entre os componentes do sistema e as políticas de manutenção elaboradas.

Por outro lado, os modelos baseados em estado representam o comportamento do sistema, seus estados e ocorrências de eventos (Maciel et al. 2011). Estes modelos são mais adequados para modelar interações complexas entre os componentes, tais como mecanismos baseados em redundância dinâmica. No entanto, os modelos baseados em estado sofrem com explosão de espaço de estado.

Diagramas de Blocos de Confiabilidade (RBD) (Kuo & Zuo 2003), Árvores de Falhas (FT) (Vesely & Roberts 1987) e Gráficos de Confiabilidade (RG) (Sahner & Puliafito 1996a) são modelos combinatórios representativos, Cadeias de Markov, bem como Redes de Petri Estocásticas (SPN) são modelos proeminentes baseados em estado. Estes modelos também são muito apropriados para a estimativa de métricas de desempenho e performabilidade.

3.4 Sistemas Coerentes

Um componente C é irrelevante para o desempenho do sistema S, se o estado do sistema não é afetado pelo estado deste componente (Kuo & Zuo 2003). Matematicamente, um componente $i(1 \leq i \leq n)$ é irrelevante para a função estrutural ϕ, se e somente se $\phi(1_i,x) = \phi(0_i,X)$ para qualquer estado do componente vetor x. Caso contrário, o componente é dito relevante.

Se um componente é relevante para um sistema, isso significa que existe pelo menos um componente de vetor de estado x de tal modo que o estado do componente i determina o estado do sistema. Dessa forma, quando outros componentes encontram-se em um certo estado (operacional ou defeituoso), especificado por $(x_1,x_2,...,x_i-1,x_i+1,...,x_n)$, o valor de $\phi(x_1,x_2,...,x_n)$ é igual a x_1. A partir dessas condições, quando o componente i funciona, o sistema funciona; quando o componente i falha, o sistema apresenta um defeito (Kuo & Zuo 2003).

Uma particularidade relevate é que melhorar o desempenho de um componente, normalmente não piora o desempenho do sistema (Kuo & Zuo 2003). Por conseguinte, a troca de um componente falho em um sistema funcionando normalmente não faz o sistema falhar. Se trocar um componente falho em um sistema falho não necessariamente recupera o sistema pois, pode haver outros componentes falhos no sistema, que o impeçam de funcionar. Usualmente, presume-se que a função de estrutura de todo sistema é uma função não decrescente do estato de todos os componentes (Kuo & Zuo 2003).

Assim, para um sistema coerente, dados dois vetores, cada um com n elementos, $\mathbf{x}$ e $\mathbf{y}$, pode-se escrever $\mathbf{x} < \mathbf{y}$ se $x_i < y_i$ para cada i e $x_i < y_i$ para pelo menos um $i(1 \leq i \leq n)$. No entanto, pode-se dizer que o vetor $\mathbf{x}$ é menor que o vetor $\mathbf{y}$ (Kuo & Zuo 2003).

Fundamentado nesta definição, segundo (Kuo & Zuo 2003), um sistema coerente satisfaz as seguintes condições:

1. $\phi(\mathbf{0}) = 0$, que dizer, o sistema é defeituoso quando todos os componentes são falhos;

2. $\phi(\mathbf{1}) = 1$, que dizer, o sistema funciona quando todos os componentes funcionam;

3. se $x < y$, então $\phi(x) \leq \phi(y)$, que dizer, a melhoria de qualquer componente não degrada o desempenho do sistema;

4. Para todos os componentes i, existe um vetor de estados de componentes em que o estado do componente i dita o estado do sistema.

3.4.1 Funções Estruturais

Funções estruturais são funções matemáticas discretas utilizadas para indicar o relacionamento entre o estado de funcionamento dos componentes de um sistema com o estado de funcionamento do sistema (Kuo & Zuo 2003, Maciel et al. 2011).

Suponha num sistema S composto por um conjunto de componentes $C = c_i | 1 \leq i \leq n$, onde o estado do sistema S e seus componentes podem estar operacionais ou falhos e n é o número de componentes do sistema. Ora, x_i é uma variável aleatória discreta, indicando o estado do componente i, assim:

$$x_i = \begin{cases} 0 & \text{se o componente } i \text{ está falho} \\ 1 & \text{se o componente } i \text{ está funcionando} \end{cases} \tag{3.9}$$

Eventualmente, um vetor $\mathbf{x} = (x_1, x_2, ..., x_n)$, representa o estado de todos os componentes do sistema. O estado do sistema é determinado pelos estados dos componentes. A função de estrutura, $\phi(x)$, mapeia o vetor do sistema $\mathbf{x}$ para 1 ou 0, como apresentado abaixo:

$$\phi(x) = \begin{cases} 0 & \text{se o sistema está falho} \\ 1 & \text{se o sistema está funcionando} \end{cases} \tag{3.10}$$

Ademais, se o estado de todos os componentes do sistema é conhecido, então o estado do sistema também é conhecido. Desta forma, o estado do sistema é uma função determinística do estado de todos os componentes. Logo,

$$\phi = \phi(x) = \phi(x_1, x_2, ..., x_n) \tag{3.11}$$

onde $\phi(x)$ é a função estrutural do sistema.

As regras de formação das funções estruturais, para componentes em série e em paralelo são mostradas a seguir:

Componentes em série:

Sejam n componentes $x_1, x_2, ..., x_n$ em série, a função estrutural ϕ desses componentes é representada a seguir:

$$\phi(x) = 1 - \prod_{i=1}^{n}(x_i) = min(x_1, x_2, ..., x_n) \tag{3.12}$$

Componentes em paralelo:

Sejam n componentes $x_1, x_2, ..., x_n$ em paralelo, a função estrutural ϕ desses compo-

nentes é representada a seguir:

$$\phi(x) = 1 - \prod_{i=1}^{n}(1 - x_i) = max(x_1, x_2, ..., x_n) \tag{3.13}$$

3.4.2 Funções Lógicas

Funções lógicas têm como objetivo indicar uma relação entre o estado dos componentes e o estado do sistema. A função lógica de um sistema coerente pode ser adotada para simplificar funções do sistema através de álgebra booleana. Todavia, em alguns casos, não é fácil simplicar funções estruturais para uma forma minimizada (Kuo & Zuo 2003).

Componentes em série:

Seja $x = (x_1, x_2, ..., x_n)$, o vetor que representa o estado de n componentes de um sistema. A função lógica em série $(Sserial)$ é definida pela equação:

$$Sserial(x) = (x_1 \wedge x_2 \wedge ... \wedge x_n) \tag{3.14}$$

Componentes paralelos:

Seja $x = (x_1, x_2, ..., x_n)$, o vetor que representa o estado de n componentes de um sistema. A função lógica paralela $(Sparalelo)$ é definida pela equação:

$$Sparalelo(x) = (x_1 \vee x_2 \vee ... \vee x_n) \tag{3.15}$$

Para obter mais detalhes sobre funções lógicas e estruturais o leitor pode recorrer a Kuo & Zuo (2003), Maciel et al. (2011).

3.5 Avaliação de Performabilidade

A modelagem de desempenho é uma abordagem estruturada para avaliar o desempenho do sistema. No entanto, a modelagem de dependabilidade lida com a representação de mudanças na estrutura do sistema devido a falhas, as quais podem afetar a disponibilidade do sistema (Jawad & Johnsen 1995).

Estes modelos são geralmente avaliados separadamente, mas uma questão surge como o desempenho depende da dependabilidade. Assim, combinar aspectos de desempenho e dependabilidade é importante para uma avaliação completa.

Avaliações separadas de desempenho e dependabilidade podem produzir uma avaliação parcial da qualidade do serviço (Meyer 1992). No entanto, há falhas estuturais que

reduzem a qualidade do serviço, sem causar falhas no sistema, ou seja, o desempenho do sistema é degradável.

A modelagem combinada de desempenho e dependabilidade é denominada performabilidade (Araújo & et al. 2011), uma medida composta que descreve uma degradação do desempenho do sistema, devido à ocorrência de eventos de falha, mesmo em decorrência delas, o sistema continuará funcionando, mas com degradações no nível de desempenho. Para realizar a avaliação de performabilidade, é comum a adoção de modelagem hierárquica para a combinação de um modelo de dependabilidade de alto nível e um modelo de desempenho de baixo nível.

A avaliação de performabilidade surgiu a partir da necessidade de se determinar a qualidade global do sistema, relacionando aspectos de desempenho e dependabilidade, como resultado de uma única avaliação.

A modelagem hierárquica tem como finalidade evitar os problemas de *largeness* e *stiffness* (Sousa & et al. 2012). O *largeness* é consequência do tamanho do espaço de estados do modelo e o *stiffness* é consequência das diferentes ordens de magnitude entre os tempos das atividades de reparo. O *stiffness* pode provocar sérios problemas durante a solução analítica do modelo, mesmo que o modelo não tenha um grande espaço de estados por causa dos diferentes tempos associados as transições temporizadas (Sousa & et al. 2012).

A avaliação de performabilidade pode ser considerada como uma das mais adequadas abordagens para entender o significado da eficácia e desempenho geral de um sistema.

3.5.1 Modelos

Existem vários tipos de modelos que podem ser utilizados para a modelagem e avaliação de performabilidade. Por exemplo, Diagramas de Bloco de Confiabilidade, Cadeias de Markov e Redes de Petri Estocásticas. Consequentemente, esses modelos têm sido utilizados para modelagem hierárquica que combina os resultados de modelos de dependabilidade e desempenho (Araújo & et al. 2011).

Para uma avaliação de performabilidade utilizando modelagem hierárquica, pode-se combinar um modelo de dependabilidade de alto nível (por exemplo, Diagramas de Bloco de Confiabilidade) que representam um sistema, e modelos de desempenho de baixo nível (por exemplo, Cadeias de Markov), que representam alguns subsistemas do modelo de dependabilidade.

3.6 Censura

Análise de sobrevivência é definida como uma técnica estatística para o estudo de dados de tempos de vida de um indivíduo, item ou componente. Dessa forma, a análise de sobrevivência permite estudar tempos de vida, também designados por tempos de sobrevivência (falhas). A característica fundamental é a presença de observações incompletas do tempo de sobrevivência chamado de censura. Para alguns indivíduos pode não ser possível observar o acontecimento de interesse durante o período em que estiverem em observação.

Um problema comum na geração de dados de confiabilidade é a censura. A censura (Ebeling 2005) ocorre quando não é possível observar o tempo de vida de um indivíduo, item ou componente, durante um período de tempo. A censura é aplicada por diversos motivos, por exemplo, quando a falha de um componente ocorre fora do período de tempo em estudo.

A censura pode ser classificada da seguinte forma (Ebeling 2005):

1. Censura à esquerda: ocorre se o evento de interesse já aconteceu quando o indivíduo foi observado.

2. Censura à direita: ocorre quando o tempo de falha é superior ao tempo observado. A seguir são descritos os tipos da censura à direita:
   i. Censura do tipo 1: o estudo terminará após um período pré estabelecido de tempo, por exemplo, suponha que um sistema é composto por 8 componentes, e que os componentes podem falhar. Foram observados os tempos de falhas dos componentes, sendo que o estudo terminou após 3 meses;
   ii. Censura do tipo 2: o estudo terminará após ter ocorrido o evento de interesse em um número pré estabelecido de indivíduos, por exemplo, o estudo é terminado depois de um número fixo de falha dos componentes.

3.6.1 Técnica de Kaplan-Meier

Problemas com dados censurados surgem com bastante frequência em estudos de confiabilidade. Estimativa da função de confiabilidade geralmente é motivo de preocupação. O estimador de função de Confiabilidade Kaplan-Meier é frequentemente utilizado quando se lida com dados censurados.

A técnica de Kaplan-Meier (Ebeling 2005, Goel et al. 2010), também conhecida como o estimador de limite-produto, pode ser utilizada para calcular os valores de confi-

abilidade não-paramétrico para conjuntos de dados com várias falhas. A técnica é muito popular para derivar uma função de confiabilidade empírica.

Assumindo que não há laços em tempos de falha e que os tempos de censura não coincidem com tempos de falha, o estimador de limite de produto Kaplan-Meier é apresentado pela Equação 3.16 a seguir:

$$\hat{R} = \prod_{\{j:t_j \leq t\}} (1 - \frac{d_j}{n_j}) \tag{3.16}$$

Onde t_j é o tempo do estudo no ponto j, d_j é o número de falhas até o ponto j e n_j é o número de indivíduos em risco antes de t_j. $\hat{R}$ é baseado na probabilidade de que um indivíduo sobrevive no final de um intervalo de tempo, com a condição de que o indivíduo estava presente no início do intervalo de tempo. $\hat{R}$ é o produto dessas probabilidades condicionais.

Como exemplo, considere os dados tempos de falha (em meses) de alguns componentes, contidos na Tabela 3.1, onde o estudo terminou quando 7 componentes haviam falhado. Como pode ser observado o número 1 indica que o componente falhou e o número 0 indica censura. As observações censuradas são marcadas por 0.

Tabela 3.1: Dados dos Tempos de Falhas.

Tempo	Status
55	1
61	0
74	1
81	1
93	0
122	0
138	1
151	1
168	1
202	0
220	0
238	1

A partir dos dados disponíveis, as estimativas de confiabilidade de Kaplan-Meier, são obtidas. A Tabela 3.2 apresenta as estimativas.

Com base nas estimativas de confiabilidade, o gráfico de Estimador de Kaplan-Meier é construído, no qual, $R(t)$ em função de t, em forma de escada. A Figura 3.5 mostra o gráfico de Estimador de Kaplan-Meier.

Tabela 3.2: Estimativas de Confiabilidade de Kaplan-Meier.

Tempo	Número de Risco	Número de Falhas	Probabilidade de Sobrevivência $\hat{R}(t)$
55	12	1	0.916667
74	10	1	0.825000
81	9	1	0.733333
138	6	1	0.611111
151	5	1	0.488889
168	4	1	0.366667
238	1	1	0.000000

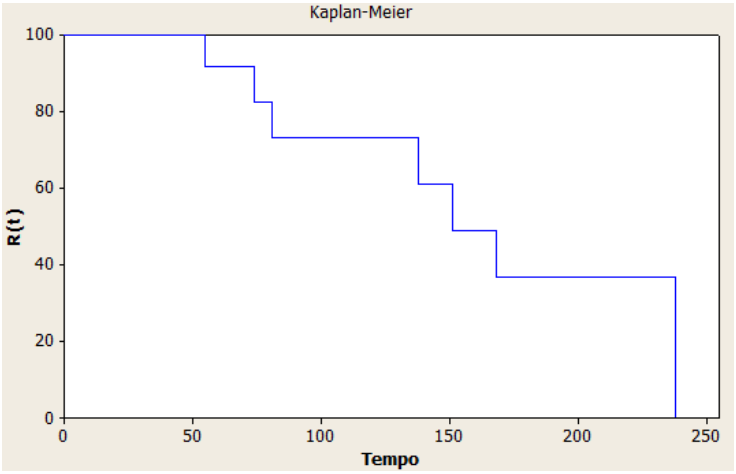

Figura 3.5: Estimador de Kaplan-Meier.

3.6.2 Teste Kolmogorov-Smirnov

O teste de Kolmogorov-Smirnov (teste K-S) (Ebeling 2005) é um teste não paramétrico para a igualdade de distribuição de probabilidade contínua, de uma dimensão que pode ser usado para comparar a amostra (valores observados) com uma distribuição de probabilidade de referência (uma amostra de teste K-S), ou para comparar duas amostras (duas amostras de teste K-S). A estatística de K-S, quantifica uma distância entre a função de distribuição empírica da amostra e a função de distribuição cumulativa da distribuição de referência, ou entre as funções de distribuição empírica de duas amostras.

Dada uma variável aleatória X, o teste K-S tem por base a análise da proximidade ou do ajustamento entre a função de distribuição empírica ou da amostra, $S(x)$, e a função de distribuição populacional, $F(x)$, que é admitida em H_0. Para uma amostra de tamanho n, a função $S(x)$ representa a soma das frequências relativas dos dados com valores menores ou iguais a x, um valor qualquer x da variável X.

Seja $(X_1, X_2, ..., X_n)$ uma amostra aleatória de uma população contínua X e $X_1, X_2, ...X_n$ a respectiva amostra ordenada, e define a função distribuição empírica $S(x)$ como se-

gue (Esteves & Sousa 2007):

$$S_(x)\begin{cases} 0, & x < x_1 \\ k/n, & x_k \leq x < x_k + 1(\text{k= 1,2,...,n-1}) \\ 1, & x \geq x_n \end{cases}$$

A função de distribuição empírica $S(x)$ é uma função em degrau que cresce $1/n$ nos pontos de salto (estatísticas ordinais da amostra).

A estatística de teste, que se denota por D_n (que é uma variável aleatória), corresponde ao supremo (ou máximo) da diferença, em valor absoluto, entre $S(x)$ e $F(x)$, quando são considerados todos os valores possíveis de X. como segue:

$$D_n = \max_x |F(x) - S(x)| \tag{3.17}$$

É possível demonstrar que, se a amostra é aleatória e provém de uma distribuição contínua conhecida, a estatística D_n só depende da dimensão da amostra, n, sendo irrelevante a forma da função distribuição da população, $F(x)$.

O teste K-S pode ser modificado para servir como um teste de goodness-of-fit (Ebeling 2005). No caso especial de testes para a normalidade de distribuição, as amostras são normalizadas e comparadas com uma distribuição normal padrão. Assim, é equivalente à configuração da média e da variância da distribuição de referência iguais para as estimativas da amostra, sabe-se que a utilização destes para definir a distribuição de referência específica altera a distribuição nula da estatística de teste.

3.7 Diagramas de Blocos de Confiabilidade

Diagrama de Blocos de Confiabilidade (RBD) (Rausand & Høyland 2004, Kuo & Zuo 2003) é uma técnica combinatorial utilizada para calcular a confiabilidade de um sistema. Normalmente, os RBDs proporcionam uma representação gráfica dos componentes do sistema e conectores, que podem ser adotados para determinar o estado geral do sistema, dado o estado de seus componentes.

Um diagrama de blocos de confiabilidade representa a relação lógica entre o funcionamento do sistema e do funcionamento dos seus componentes (Kuo & Zuo 2003). Os modelos RBDs são representados por conjuntos de blocos (denotados como retângulos) que indicam os componentes em que os arcos definem a relação lógica.

O diagrama de blocos de confiabilidade é utilizado, particularmente, em sistemas modulares que consistem de muitos módulos independentes, onde cada um pode ser

representado por um bloco de confiabilidade.

Embora o RBD tenha sido inicialmente um modelo proposto para calcular a confiabilidade de sistemas, pode ser utilizado para calcular outras métricas de dependabilidade, tais como: disponibilidade e manutenabilidade.

Em RBD, é possível representar um componente físico no modo operacional por um bloco, estimar a confiabilidade de cada bloco individualmente. Portanto, para representar uma falha de um componente, é necessário remover o bloco correspondente ao componente do modelo. Se existir pelo menos um caminho que faça a ligação entre os blocos, o sistema continua funcionando corretamente. Assim, se removida uma quantidade suficiente de blocos para interromper a conexão entre os blocos, o sistema falha (Kuo & Zuo 2003). No entanto, modelos RBD são impróprios para modelagem de dependências de falhas e reparação que são frequentemente encontrados na representação de políticas de manutenção e mecanismos redundantes, particularmente aquelas baseadas em métodos de redundância dinâmica.

Os modelos RBDs têm sido usados para representar arranjo em série, arranjo em paralelo, arranjo K-out-of-n e arranjo bridge.

Os parágrafos seguintes descrevem os arranjos adotados neste trabalho.

Arranjo em Série. A Figura 3.6 representa o modelo RBD, em que sempre que um componente falhar, então todo o sistema também falha. Supondo um sistema com n componentes, disponibilidade/confiabilidade (P_s) é estimada da seguinte forma

$$P_s = \prod_{i=1}^{n} p_i \tag{3.18}$$

em que p_i refere-se ao componente i disponibilidade/confiabilidade.

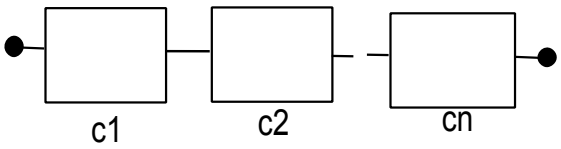

Figura 3.6: Arranjo em Séries.

Arranjo em Paralelo. A Figura 3.7 representa o modelo RBD. Um sistema está em estado de falha somente quando todos os componentes falharem. Levando-se em conta n componentes, disponibilidade/confiabilidade é calculada utilizando

$$P_s = 1 - \prod_{i=1}^{n} (1 - p_i) \tag{3.19}$$

em que p_i refere-se ao componente i disponibilidade/confiabilidade.

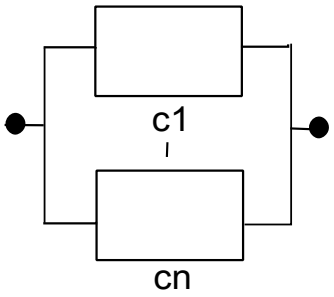

Figura 3.7: Arranjo em Paralelo.

Arranjo K-out-of-n. A Figura 3.8 representa o modelo RBD, em que é necessário um número mínimo de componentes (k) a fim de manter o sistema operacional. Supondo um sistema com n componentes idênticos, a disponibilidade/confiabilidade (P_s) é estimada utilizando

$$P_s = \sum_{i=k}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) p^i (1-p)^{n-i} \tag{3.20}$$

em que p refere-se ao componente disponibilidade/confiabilidade.

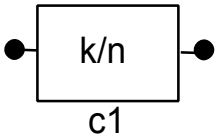

Figura 3.8: Arranjo K-out-of-n.

Um modelo RBD pode adotar vários componentes com arranjos diferentes, Figura 3.9. Em tal caso, a redução e a soma dos produtos disjuntos (Maciel et al. 2011) são abordagens representativas para estimar a disponibilidade ou a confiabilidade do sistema.

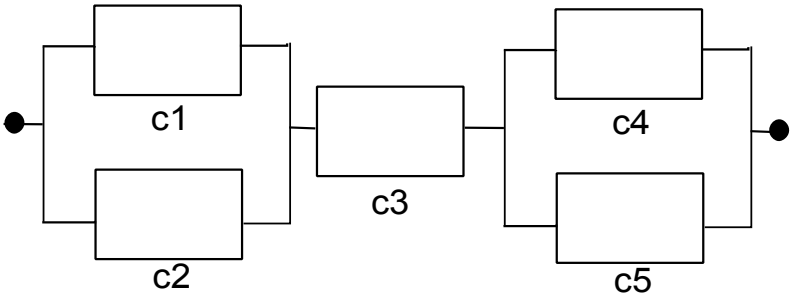

Figura 3.9: Arranjo Série-Paralelo.

3.8 Redes de Petri

O conceito de redes de Petri foi introduzido por Carl Adam Petri, no ano de 1962, com a apresentação da sua tese de doutorado "*Kommunikation mit Automaten*" (comunicação com autômatos) (Murata 1989, Maciel et al. 1996) na faculdade de Matemática e Física da Universidade Darmstadf na Alemanha. Redes de Petri (PN) (Balbo 2001) são uma família de formalismos muito bem adequada para a modelagem de diversos tipos de sistemas, desda concorrência, sincronização, mecanismos de comunicação, bem como os atrasos são naturalmente representados.

Rede de Petri é um grafo bipartido direcionado, em que lugares (representados por círculos) denotam estados locais e transições (representados como retângulos) representam ações. Arcos (arestas direcionado) conectam lugares para transições e vice-versa. Tokens (pequenos círculos preenchidos) que denotam o estado (ou seja, a marcação) de uma PN. Um arco inibidor é um tipo de arco especial que mostra um pequeno círculo branco em uma borda, em vez de uma seta, e eles representam a indisponibilidade dos Tokens nos lugares. A Figura 3.10 apresenta os elementos de rede de Petri, e, a Figura 3.11 mostra um exemplo de rede de Petri.

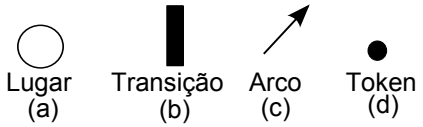

Figura 3.10: Elementos de Rede de Petri.

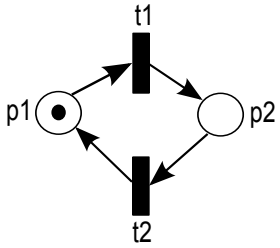

Figura 3.11: Exemplo de Rede de Petri.

A Figura 3.12 (Alves 2007) apresenta os períodos do dia. Os lugares representam os períodos dos dias (dia e noite), enquanto as transições representam os eventos que alteram o período do dia (amanhecer ou anoitecer). Neste exemplo, o arco dirigido do lugar dia para a transição anoitecer indica que, para anoitecer, é necessário que haja um *token* no lugar dia. De maneira análoga, o arco dirigido do lugar noite para a transição

amanhecer indica que, para amanhecer, é necessário que haja um token no lugar noite. A localização do token na rede indicará, portanto, se é dia (Figura 3.12(a)) ou noite (Figura 3.12(b)).

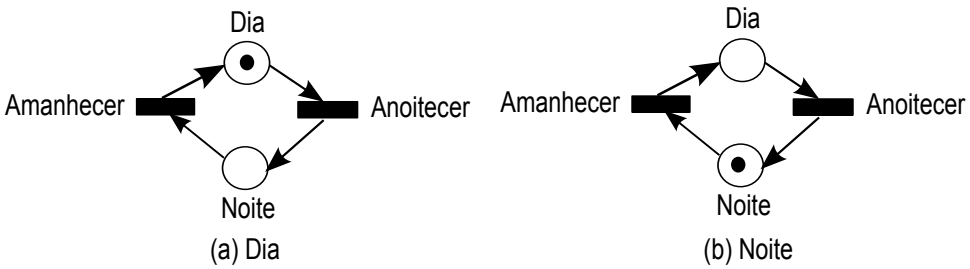

Figura 3.12: Rede de Petri Representando o Dia.

Na representação gráfica, um lugar pode ser conectado a uma transição por meio de múltiplos arcos (arcos multivalorados) que podem ser compactados em um único arco rotulado. Dessa forma, estes arcos podem ser substituídos por um único arco com um peso associado. À medida que uma transição é disparada, ela consome os *tokens* dos lugares de entrada, colocando outros *tokens* nos lugares de saída. A quantidade de *tokens* consumidos e colocados nos lugares de saída é dada pelo peso do arco que conecta os lugares a esta transição. A Figura 3.13 mostra um exemplo.

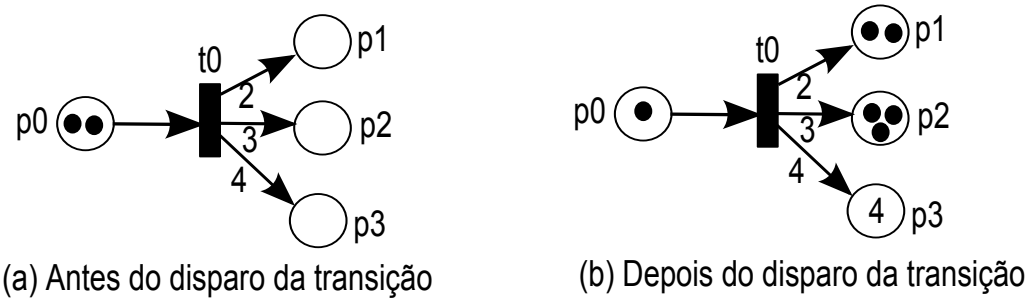

Figura 3.13: Exemplo de uma Rede de Petri com Peso nos Arcos.

A representação formal de um modelo PN é a quíntupla $PN = (P, T, F, W, M_0)$, onde:

- P é o conjunto finito de lugares;
- T é o conjunto finito de transições, $P \cap T = \emptyset$;
- $F \subseteq (P \times T) \cup (T \times P)$ é o conjunto de arcos;
- $W : F \rightarrow IR^+ \cup \{0\}$ é a função de atribuição de peso aos arcos;
- $M_0 : P \rightarrow IN$ é a função de marcação inicial.

3.8.1 Rede de Petri Marcada

Uma marca (*token*) é um conceito primitivo em PN, da qual lugar e transição. Os *token* são informações atribuídas aos lugares. Uma marcação associa um k (inteiro não-negativo) a cada lugar da rede. A seguir são apresentadas as seguintes definições formais: marcação, vetor de marcação e rede de Petri marcada.

- **Marcação**: Seja P o conjunto de lugares de uma PN. Define-se formalmente marcação como uma função que mapeia o conjunto de lugares P a inteiros não negativos $M : P \rightarrow \mathbb{N}$.
- **Vetor Marcação**: Seja P o conjunto de lugares de uma PN. A marcação pode ser definida formalmente como um vetor $M = (M(p_1), ..., M(p_n)))$, no qual $n = |P|$, para todo $p_i \in P$, tal que $M(p_i) \in \mathbb{N}$.
- **Rede Marcada**: Define-se uma rede de Petri marcada pela dupla $RM(R;M_0)$, no qual R é a estrutura da rede e M_0 é a marcação inicial.

3.8.2 Grafo de Alcançabilidade

Usualmente, um grafo rotulado e direcionado é adotado para mostrar todas as possíveis marcações que a Rede de Petri pode alcançar. Este grafo é usualmente chamado de grafo de alcançabilidade.

Podemos definir um grafo de alcançabilidade como sendo uma tupla (V,E), onde V representa o conjunto de vértices representados pelas marcações possíveis, e E é o conjunto de arestas rotuladas.

Exemplificando, considere $M = \{m0 = |1,0,0|, m1 = |0,1,0|, m2 = |0,0,1|\}$ sendo o conjunto das marcações alcançáveis da rede de Petri representada na Figura 3.14. O respectivo grafo de alcançabilidade é representado na Figura 3.15.

3.8.3 Redes Elementares

As redes elementares são blocos básicos que permitem a modelagem de sistemas mais complexos. A seguir serão mostradas algumas das redes elementares, tais como: sequência, distribuição, junção, escolha não-determinística, atribução e confusão.

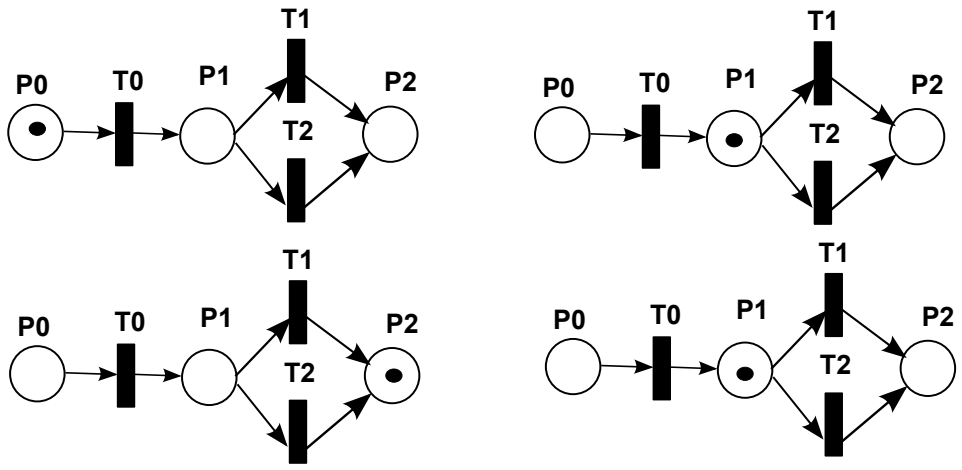

Figura 3.14: Exemplo de Rede de Petri

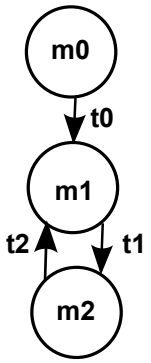

Figura 3.15: Exemplo de Grafo de Alcançabilidade

3.8.3.1 Sequência

A sequência é uma rede que permite a representação de ações consecutivas, desde que uma dada condição seja satisfeita (Maciel et al. 1996). Ou seja, após a execução de cada ação, uma nova condição poderá ser disparada, permitindo, assim, a execução de uma nova ação. A Figura 3.16 mostra um exemplo dessa rede, onde um *token* no lugar P0 habilita a transição T0, e com o disparo dessa transição uma nova condição é estabelecida (P1 é marcado). Assim, com um *token* no lugar P1 habilita a transição T1, consequentemente, com o disparo dessa transição P2 é marcada. Essa nova condição pode permitir o disparo de uma nova condição associada ao lugar P2.

3.8.3.2 Distribuição

Esta rede permite a criação de processos paralelos a partir de um processo hierarquicamente superior (Maciel et al. 1996). Como mostrado na Figura 3.17, o disparo da

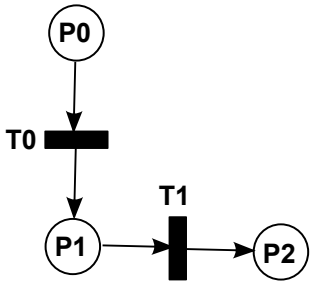

Figura 3.16: Sequência

transição T0 adiciona um *token* no lugar P1 e outro no lugar P2. Essas novas condições (P1 e P2) permitem a execução de novos processos paralelos.

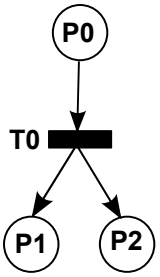

Figura 3.17: Distribuição

3.8.3.3 Junção

Esta rede permite a modelagem da sincronização de processos paralelos (ver Figura 3.18). Ela combina duas ou mais redes, deixando que outro processo continue sua execução somente após o término de todos os processos paralelos que o antecedem (Maciel et al. 1996). Como apresentado na Figura 3.18, a transição T0 estará habilitada, se ambas as pré-condições contiverem *tokens* (P0 e P1). Se essa condição for satisfeita, então a transição T0 poderá ser disparada, retirando um *token* dos lugares P0 e P1 e colocando em P2.

3.8.3.4 Escolha Não-Determinística

A seguir, é apresentada uma rede elementar que pode ser denominada de conflito, escolha ou decisão, dependendo da aplicação (Maciel et al. 1996). A Figura 3.19 mostra a escolha não determinística, onde o disparo de uma transição desabilita o disparo de uma outra transição. Existindo um *token* em P0 , T0 e T1 torna-se conflitante, isto é, o disparo de uma transição elimina a possibilidade da outra.

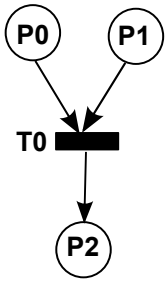

Figura 3.18: Junção

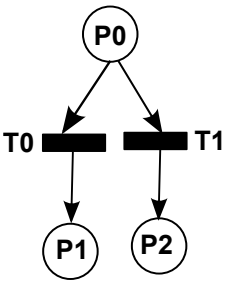

Figura 3.19: Escolha

3.8.3.5 Atribuição

Atribuição é uma rede elementar que permite que dois ou mais processos habilitem um terceiro processo (Maciel et al. 1996). Na Figura 3.20, tanto a transição T0 quanto a transição T1 são independentes, porém ambas têm um lugar de saída em comum. Desse modo, após o disparo de qualquer uma dessas transições, cria-se uma condição (P2 é marcado) que possibilita o disparo de uma outra transição.

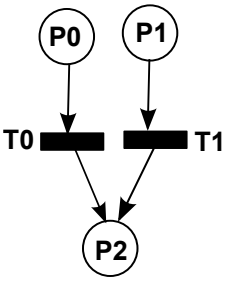

Figura 3.20: Atribuição

3.8.4 Propriedades das Redes de Petri

O estudo das propriedades de redes de Petri permite a análise do sistema modelado. Os tipos de propriedades podem ser divididos em duas categorias: as propriedades de-

pendentes de marcação inicial, conhecidas como propriedades comportamentais, e as propriedades não dependentes de marcação, conhecidas como propriedades estruturais (Murata 1989).

3.8.4.1 Propriedades Comportamentais

As propriedades comportamentais são aquelas que dependem apenas da marcação inicial da rede de Petri. As propriedades abordadas são: alcançabilidade, limitação, segurança, vivacidade e cobertura (Maciel et al. 1996).

- **Alcançabilidade**: indica a possibilidade de uma determinada marcação ser atingida pelo disparo de um número finito de transições a partir de uma marcação inicial. Uma marcação M_0 é dita alcançavel l a partir de M', se existir uma sequência de disparo que transforme M_0 em M'. A sequência de disparo é descrita pelo conjunto $S = t1, t2, ..tn$. Nesse caso, M' é alcançavel a partir de M_0 por S. No qual S é formalmente descrito por $M_0[S > M'$.

- **Limitação**: o limite k é o número máximo de marcas que um lugar pode acumular. Uma rede de Petri marcada $RM = (R; M_0)$ é k-limitada se o número de marcas de cada lugar de RM não exceder k em qualquer marcação acessível de $RM(max(M(p)) = k, \forall p \in P)$.

- **Segurança**: é uma especialização da propriedade de limitação. O conceito de limitação descreve que um lugar p_i é k-limitado se o número de marcas que esse lugar pode acumular estiver limitado ao número k. Um lugar que é 1-limitado pode ser simplesmente chamado de seguro.

- **Vivacidade**: está definida em função das possibilidades de disparo das transições. Uma rede é considerada *live* se, independentemente das marcações que sejam alcançáveis a partir de M_0, for sempre possível disparar qualquer transição da rede através de uma sequência de transições $L(M_0)$. A ausência de bloqueio (*deadlock*) em sistemas está fortemente ligada ao conceito de vivacidade, pois *deadlock* em uma rede de Petri é a impossibilidade do disparo de qualquer transição da rede. O fato de um sistema ser livre de *deadlock* não significa que seja *live*, entretanto um sistema *live* implica um sistema livre de *deadlocks*.

- **Cobertura**: A propriedade de cobertura está conectada ao conceito de alcançabilidade e *liveness*. Quando se deseja saber se alguma marcação M' pode ser obtida a partir de uma marcação. Uma marcação M' é dita coberta se existe uma marcação M'' tal que $M'' > M'$.

3.8.4.2 Propriedades Estruturais

As propriedades estruturais são aquelas que dependem apenas da estrutura da rede de Petri. Essas propriedades refletem características independentes de marcação. As propriedades analisadas neste trabalho são: limitação estrutural, conservação, repetitividade e consistência (Maciel et al. 1996).

- **Limitação Estrutural**: uma rede de Petri $PN = (P, T, F, W, M_0)$ é classificada como estruturalmente limitada se for limitada para qualquer marcação inicial.
- **Conservação**: a conservação é uma considerável propriedade das PN, pois permite a verificação da não destruição de recursos através da conservação de *tokens*.
- **Repetitividade**: uma rede é classificada repetitiva se existe uma marcação e uma sequência de transições disparáveis, em que as transições dessa rede são disparadas sempre.
- **Consistência**: uma rede é dita consistente se dada uma sequência de transições disparáveis a partir de uma marcação inicial $M0$ retorna a $M0$, contudo todas as transições da rede são disparadas pelo menos uma vez.

3.9 Rede de Petri Estocástica

Rede de Petri estocástica (SPN) é uma das extensões proeminentes de rede de Petri (PN) (Balbo 2001) utilizada para a modelagem de desempenho e dependabilidade de sistemas. Uma rede de Petri estocástica adiciona tempo ao formalismo de redes de Petri, com a diferença de que os tempos associados às transições temporizadas são distribuídos exponencialmente, enquanto o tempo associado às transições imediatas é zero. As transições temporizadas modelam atividades através dos tempos associados, de modo que o período de habilitação da transição temporizada corresponde ao período de execução da

atividade, e o disparo da transição temporizada corresponde ao término da atividade. Níveis diferentes de prioridade podem ser atribuídos às transições. A prioridade de disparo das transições imediatas é superior à das transições temporizadas. As prioridades podem solucionar situações de confusão (Marsan et al. 1998). As probabilidades de disparo associadas às transições imediatas podem solucionar situações de conflito (Balbo 2001, Marsan et al. 1998).

Formalmente, uma rede de Petri estocástica (SPN) é um grafo direto bipartido representado por uma tupla $SPN = (P, T, \pi, I, O, H, M_0, W, G)$, em que (Balbo 2001):

- P é o conjunto de lugares;
- $T = T_{imm} \cup T_{timed}$ é o conjunto de transições imediata (T_{imm}) e temporizada (T_{timed}) , de tal modo que $T \cap P = \emptyset$;
- $\pi : T \to \mathbb{N}$ é a função de prioridade, de tal modo que
$$\pi(t) = \begin{cases} \geq 1, & if(t \in T_{imm}) \\ 0, & if(t \in T_{timed}) \end{cases}$$
- $I, O, H : T \to Bag(P)$ são as funções de entrada, de saída e de inibição, respectivamente, em que $Bag(P)$ é o multiconjunto em P ($Bag(P) : P \to \mathbb{N}$);
- $M_0 : P \to \mathbb{N}$ é a função de marcação inicial;
- $W : T \to \mathbb{R}$ é a função peso, que mapeia uma transição imediata para um peso e uma transição temporizado na respectiva taxa λ_t:
$$W(t) = \begin{cases} w_t \geq 0, & if(t \in T_{imm}) \\ \lambda_t > 0, & if(t \in T_{timed}) \end{cases}$$
- $G \in (\mathbb{N}^{|P|} \to \{true, false\})^{|T_{imm}|}$ é um vetor que atribui uma condição de guarda relacionada à marcação do lugar a cada transição imediata.

Os modelos SPN apresentam dois tipos de estados (marcações), os estados tangíveis (*tangible*) e os estados voláteis (*vanish*) (Balbo 2001). Os estados voláteis são criados em decorrência da marcação dos lugares que são pré-condições de habilitação de uma ou mais transição imediata. O termo *vanish* é utilizado porque as marcações chegam a esses lugares e são imediatamente consumidas. O tempo de permanência das marcações nesses lugares é zero. Os estados tangíveis são criados em consequência da marcação dos lugares que são pré-condições de habilitação de uma transição temporizada (Marsan et al. 1998).

As transições temporizadas são caracterizadas por diferentes semânticas de disparo denominadas como *single server*, *multiple server* e *infinite server* (Marsan et al. 1998). Neste trabalho é utilizada a semântica *single server*.

Na semântica *single server*, as marcações são processadas serialmente. Depois do primeiro disparo da transição temporizada, o temporizador é reiniciado como se a transição temporizada tivesse sido habilitada novamente. Esse tipo de semântica é usada nos modelos de disponibilidade, considerando-se que haja unicamente uma equipe de manutenção, quando diversos componentes do sistema entram numa condição de falha.

Modelos SPN possibilitam a geração de grafos de alcançabilidade a partir dos quais cadeias de Markov de tempo contínuo (CTMC) são diretamente oriundas (Marinho 2010). A Figura 3.21 mostra um exemplo de geração do grafo de alcançabilidade a partir de um modelo SPN. No modelo SPN mostrado na Figura 3.21 (a), existe um conflito entre duas transições imediatas ($T1$ e $T2$). A Figura 3.21 (b) apresenta o grafo de alcançabilidade com a indicação de que o estado $P1$ é volátil. O disparo da transição temporizada $T0$ torna o lugar $P1$ marcado, habilitando as duas transições imediatas, $T1$ e $T2$, gerando o estado $P1$. Há uma mudança imediata (tempo zero) para o estado $P2$ ou $P3$, através do disparo da transição imediata $T1$ ou $T2$, com probabilidades $\frac{\alpha}{\alpha+\beta}$ e $\frac{\beta}{\alpha+\beta}$, respectivamente (Marinho 2010). A Figura 3.21 (c) apresenta o grafo de alcançabilidade tangível após a eliminação do estado volátil $P1$.

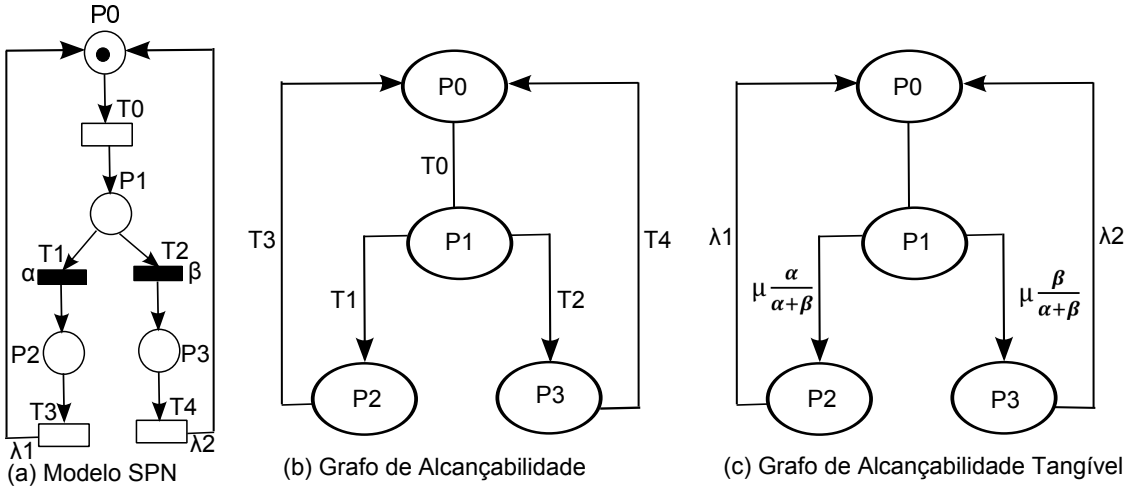

Figura 3.21: Geração de Grafo de Alcançabilidade.

A taxa na qual o sistema se move do estado $P0$ para $P2$ ou $P3$ é obtida pelo produto da taxa λ da transição do estado $P0$ para o estado volátil $P1$, com a probabilidade de ir do estado $P1$ para o estado $P2$ ou $P3$.

Redes de Petri estocásticas marcadas, com um número finito de lugares e transições,

são isomórficas às cadeias de Markov (Murata 1989). O isomorfismo de um modelo SPN com uma cadeia de Markov é obtido a partir do grafo de alcançabilidade reduzido, que é dado através da eliminação dos estados voláteis e rótulo dos arcos com as taxas das transições temporizadas e pesos das transições imediatas. As medições de desempenho e dependabilidade são obtidas através de simulações e análises em estado estacionário e transiente, baseadas na cadeia de Markov, embutida no modelo SPN (Bolch et al. 2006).

Os modelos SPN são utilizados para avaliação de dependablidade, desempenho e performabilidade de sistemas, visto que permitem a descrição das atividades de sistemas através de grafos de alcançabilidade. Esses grafos podem ser convertidos em modelos Markovianos, que são utilizados para avaliação quantitativa do sistema avaliado.

De agora em diante $\#p$ denota o número de tokens no lugar p; e $P\{exp\}$ estima a probabilidade da expressão exp.

3.10 *Phase-Type Distributions*

Embora SPN assuma a distribuição exponencial para transições temporizadas, as atividades não-exponenciais podem ser representadas utilizando as *phase-Type distributions*. Basicamente, são adotadas diferentes combinações de lugares, transições imediatas e temporizadas para representar diferentes atrasos de distribuições.

A técnica de aproximação por fases (Watson & A. 1991) tem sido normalmente utilizada para representar o comportamento de distribuições desconhecidas, que, a partir da média de atraso (μ_d) e desvio padrão (σ_d), uma aproximação por fase é levada em conta, por exemplo, Erlang, Hipoexponencial e Hiperexponencial. É importante mencionar que essa técnica tem sido utilizada com sucesso na modelagem de atividades não-exponenciais. O seguinte algoritmo é considerado (Callou & et al. 2012):

- Se $\mu_d = \sigma_d$, apenas uma única transição temporizada é adotada;
- Assumindo $\mu_d/\sigma_d \in \mathbb{N}$ e $\mu_d/\sigma_d \neq 1$, a aproximação por fase considera uma sub-rede Erlang (Figure 3.22 (a)), de tal modo que $\gamma = (\frac{\mu_d}{\sigma_d})^2$ and $\lambda = \gamma/\mu_d$;
- Considerando que $\mu_d > \sigma_d$, uma sub-rede hipoexponencial é adotada (Figure 3.22 (b)) e

$$(\frac{\mu_d}{\sigma_d})^2 - 1 \leq \gamma < (\frac{\mu_d}{\sigma_d})^2 \tag{3.21}$$

$$\lambda_1 = \frac{1}{\mu_1} \; and \; \lambda_2 = \frac{\gamma}{\mu_2} \tag{3.22}$$

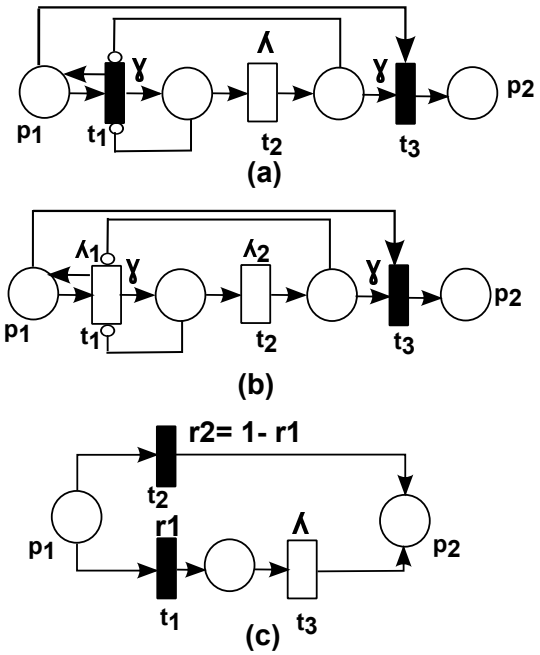

Figura 3.22: Distribuições: (a) Erlang, (b) Hipoexponencial e (c) Hiperexponencial

$$\mu_1 = \frac{\mu_d \pm \sqrt{\gamma(\gamma+1)\sigma_d{}^2 - \gamma\mu_d{}^2}}{\gamma+1} \tag{3.23}$$

$$\mu_2 = \frac{\gamma\mu_d \mp \sqrt{\gamma(\gamma+1)\sigma_d{}^2 - \gamma\mu_d{}^2}}{\gamma+1} \tag{3.24}$$

- Se $\mu_d < \sigma_d$, à aproximação assume uma sub-rede hiperexponencial (Figure 3.22 (c)), em que

$$r_1 = \frac{2\mu_d{}^2}{(\mu_d{}^2 + \sigma_d{}^2)} \tag{3.25}$$

$$r_2 = 1 - r_1 \tag{3.26}$$

$$\lambda = \frac{2\mu_d}{(\mu_d{}^2 + \sigma_d{}^2)} \tag{3.27}$$

3.11 Considerações Finais

Este capítulo apresentou os principais conceitos que envolvem essa dissertação. Primeiramente, foram descritos conceitos sobre gerencimento de riscos, incluindo as categorias de riscos e as atividades do gerenciamento de riscos. Posteriormente, foram abordados conceitos sobre avaliação de desempenho, avaliação de dependabilidade, avaliação de

performabilidade e censura. Posteriormente, foi apresentado os diagramas de bloco de confiabilidade, tais como os arranjos adotadas e uma introdução sobre redes de Petri, assim como definições, conceitos básicos e propriedades, as quais podem ser divididas em duas categorias: propriedades comportamentais e propriedades estruturais. Por fim, foram apresentadas as redes de Petri estocásticas (SPNs), que é a extenção de redes de Petri proeminentemente adotada neste trabalho.

4
Metodologia e Modelos

Seu trabalho irá tomar uma grande parte da sua vida
e o único meio de ficar satisfeito é fazer o que você
acredita ser um grande trabalho.
—STEVE JOBS

Neste capítulo é apresentada a metodologia adotada para avaliação de performabilidade de riscos de desenvolvimento em projetos de *software* além de um exemplo motivacional. Em seguida são apresentados os modelos adotados com base em diagramas de blocos de confiabilidade e redes de Petri estocásticos para estimar as métricas de dependabilidade e performabilidade.

4.1 Método Proposto

A indústria de *software* lida com vários tipos de riscos que fazem com que os projetos de desenvolvimento de *software* sejam desviados de seu planejamento original, cronograma, prazo de entrega e, consequentemente, sua qualidade. Dessa forma, é necessário gerenciar os riscos do projeto.

O gerenciamento de riscos é de grande importância para projetos de *software*, devido à inerentes incertezas relacionadas às suas execuções. Os riscos são classificados de acordo com diferentes categorias, que podem incluir os riscos do projeto, riscos de negócios e os riscos do produto. As premissas desta dissertação direcionam especial atenção aos riscos do projeto que, por exemplo, podem afetar o cronograma e recursos do projeto Guide (2013).

Geralmente, o gerenciamento de riscos considera os seguintes atividade (Somerville 2011) (Figura 4.1): (i) identificação, (ii) análise, (iii) planejamento e (iv) monitoramento.

O processo é iterativo (Somerville 2011), começa no planejamento do projeto e continua durante toda a vida do projeto. A avaliação de risco é adotada na atividade de análise, a fim de estimar o impacto quantitativo/qualitativo dos riscos identificados.

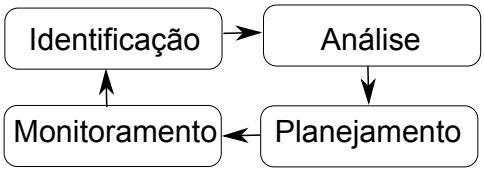

Figura 4.1: Atividades do Gerenciamento de Riscos Adaptado de (Somerville 2011).

Utilizando os conceitos de dependabilidade, os riscos no processo de desenvolvimento de *software* podem ser estimados probabilisticamente e as técnicas (por exemplo, redundância dinâmica) podem ser utilizadas para evitar ou reduzir a ocorrência de tais problemas. Por exemplo, podem ser adotados os seguintes conceitos e técnicas:

- Falha: a ocorrência de um risco (por exemplo, alguns desenvolvedores deixaram o projeto, afetando a quantidade mínima de desenvolvedores necessários para manter o projeto de *software* operacional);
- MTTF: o tempo médio para um desenvolvedor deixar o projeto ou tempo médio para um desenvolvedor falhar na implementação de um requisito/funcionalidade. Aumentando o MTTF, pode-se baixar a probabilidade de falha dos projetos de *software*;
- MTTR: o tempo médio para a substituição de um desenvolvedor ou recuperação do desenvolvedor para implementar outros requisitos. Com a diminuição do MTTR, são grandes as chances dos projetos de *software* obter sucesso;
- *Cold standby*: um desenvolvedor de *backup* está a trabalhar para um outro projeto, mas ele pode substituir um membro da equipe sempre que necessário;
- *Hot standby*: programação em pares;
- Manutenção Preditiva: reuniões periódicas com os *stakeholders* (partes interessadas) a fim de evitar falhas durante a implementação do requisito.

A Figura 4.2 apresenta o método proposto para avaliação probabilística de risco em projetos de desenvolvimento de *software*, que é realizada durante a atividade de análise (ou seja, após a identificação do risco). Nossa ênfase é na avaliação quantitativa

do risco, que é uma sub-parte da análise de risco. O método assume dados históricos como entrada e dados probabilisticos como saída. Esse método é importante para avaliar probabilisticamente os riscos de desenvolvimento em projetos de *software*. Gerentes de projetos de *software* podem aplicar esse método, para evitar ou mitigar os riscos de desenvolvimento.

A seguir, as atividades da metodologia são detalhadas.

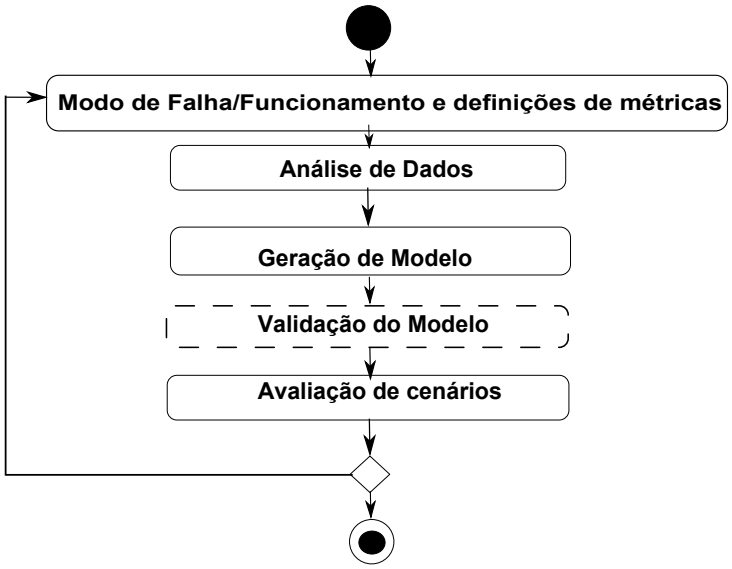

Figura 4.2: Atividades do Método Proposto.

Modo de Falha/Funcionamento e definições de métricas. Nesta atividade, os modos de falha/funcionamento de um projeto de *software* são definidos, bem como as métricas desejadas considerando cada risco identificado. Por exemplo, assumindo rotatividade de desenvolvedor, o modo de falha/funcionamento considera um mínimo de 4 desenvolvedores para manter o processo de *software* em execução, e as métricas de interesse que incluem disponibilidade e confiabilidade. O gerente de projeto seria o responsável por essa atividade, o qual, a partir de dados históricos do projeto de *software*, pode definir as métricas que serão tratadas, para cada risco identificado e consequentemente seu modo de falha/funcionamento.

Análise de Dados. Os valores médios, como MTTF e MTTR, são comumente necessários para estimar as métricas/atributos de dependabilidade. Em tal caso, um indivíduo (por exemplo, gerente do projeto) pode tomar dados históricos (a partir de projetos correntes ou outros projetos) para estimar esses valores sempre que disponíveis. Apesar dos projetos de *software* serem distintos, projetos anteriores podem fornecer dados importantes para estimar muitas métricas, dando percepções interessantes para os gerentes

de projeto em relação aos riscos identificados.

No que diz respeito às estimativas de MTTF, a censura (Seção 3) pode ocorrer no sentido de que os dados são incompletos, uma vez que nem todos os componentes falharam durante um período de observação (Ebeling 2005) (por exemplo, um desenvolvedor não deixou o projeto). Em tal situação, um método não paramétrico, de Kaplan-Meier (Seção 3), fornece uma estimativa inicial para uma distribuição empírica e, em seguida, para um encaixe de distribuição teórica é realizada e avaliada utilizando o teste estatístico (ou seja, teste K-S) (Seção 3), para avaliar o *goodness-of-fit* (Ebeling 2005). Neste trabalho, *Phase-Type Distributions* são consideradas (Seção 3), uma vez que proporcionam aproximação para distribuições gerais de probabilidade e podem ser adotadas na avaliação das CTMC e modelos SPN. Assumindo dados completo, a técnica *moment-matching* é aplicada diretamente (Watson & A. 1991). Se não há dados históricos disponíveis, MTTF e MTTR podem ser definidos com base em uma expectativa e experiência do gerente do projeto.

Geração de Modelo. Esta atividade corresponde à geração de modelos de dependabilidade e performabilidade, de acordo com a definição dos modos de falha e estimados os MTTFs/MTTRs. Neste trabalho, os modelos são construídos utilizando um modelo híbrido, que adota resultados de diferentes modelos (Seção 4.3). A abordagem considera as características do modo de funcionamento/falha, bem como a complexidade para obter uma métrica definida. Após estimar as métricas de dependabilidade, o modelo performabilidade é construído.

Validação do Modelo. Uma vez que o modelo de sistema é criado, a validação é importante para avaliar a métrica/atributo estimados pelo modelo. No entanto, essa atividade pode ser opcional, como o modelo pode representar um projeto que ainda não foi iniciado. A mesma situação pode ocorrer sempre que MTTF/MTTR são definidos pela experiência.

Por outro lado, supondo que um projeto em andamento, um indivíduo possa coletar dados e realizar uma comparação. Dependendo dos dados disponíveis e sua complexidade, o erro relativo (*err*) ou um teste estatístico pode validar o modelo (por exemplo, teste emparelhado). $err = (|v\_model - v\_data| \times 100)/|v\_data|$, em que v_model é a métrica estimada pelo modelo, e v_data é a métrica obtida por análise dos dados coletados.

Avaliação de Cenários. Uma vez que o modelo base é construído, o gestor pode conceber vários cenários para avaliar o respectivo impacto sobre a dependabilidade, modificando a estrutura e os valores no modelo atual. Após a avaliação do modelo, sempre

que as métricas estimadas não fornecem os valores esperados, as atividades anteriores podem ser executadas de forma iterativa.

O método proposto pode ser automatizado por ferramentas, no sentido em que, a partir de um modelo de alto nível (por exemplo, diagramas da UML), os modelos de dependabilidade podem ser gerados automaticamente considerando o modo de falha ou de funcionamento especificado. Da mesma forma, tal ferramenta poderia considerar uma funcionalidade integrada para estimar MTTF/MTTR a partir de um banco de dados com dados históricos. No entanto, tal ferramenta (semi-) automática está fora do escopo deste trabalho. A utilização dessa ferramenta pelos gerentes de projetos, seria viável por conta da sua familiaridade com diagramas da UML.

4.2 Exemplo Motivacional

Vamos considerar um projeto de *software* com 3 desenvolvedores e rotatividade de desenvolvedor como um risco proeminente. A Tabela 4.1 descreve o MTTF (tempo médio para um desenvolvedor deixar o projeto) e MTTR (tempo médio para a substituição de um desenvolvedor) estimados para os desenvolvedores assumindo um mês como a unidade de tempo. A partir desses valores médios, a disponibilidade pode ser calculada para um único componente/desenvolvedor (Capítulo 3) e, a partir dos componentes, a disponibilidade do sistema é estimada (Seção 4.3).

Tabela 4.1: MTTF e MTTR em Meses.

MTTF	MTTR
18,0	3,0

Um gerente pode requerer a disponibilidade do projeto considerando três condições de trabalho: (1) todos os desenvolvedores trabalhando; (2) 2 de 3; e (3) pelo menos um desenvolvedor. Tais suposições podem impactar a ordem de implementação de requisito/funcionalidade, o que pode afetar ainda mais a execução do projeto, quando um desenvolvedor o deixa. Usando os modelos descritos na Seção 4.3, a Tabela 4.2 apresenta os valores estimados, através da Equação 3.20 descrita no Capítulo 3 (Seção 3.7).

O cenário 1 tem a pior disponibilidade, o que indica que, em um mês, o projeto está operacional 62,99% ($0,6299 \times 30$, aproximadamente 18,89 dias). No entanto, o cenário 3 indica uma melhor disponibilidade ($0,9970$ ou $99,70\%$), apesar de um possível impacto no desempenho.

Tabela 4.2: Resultados dos Cenários.

Cenário	Disponibilidade
1	0,6299
2	0,9446
3	0,9970

Com base nos valores estimados, um gerente de projeto deve planejar as ações apropriadas para mitigar o risco, bem como os custos associados devido à indisponibilidade $(UA = 1 - A)$ para o cenário adotado. Como exemplo, dependendo da criticidade do projeto, o gerente pode tomar medidas para aumentar o MTTF, como bônus ou melhores salários. De modo semelhante, pode ser reduzido MTTR, assumindo um programador remanescente para continuar temporariamente a implementação. Outra alternativa é a consideração de mais desenvolvedores do projeto de *software*.

Falha de desenvolvedor pode causar degradação no desempenho. A modelagem de performabilidade pode avaliar, por exemplo, a vazão de entrega (funcionalidades entregues por unidade de tempo).

Vamos supor um processo em cascata, com as seguintes atividades: análise de requisito; projeto; implementação; teste e implantação. A Tabela 4.3 apresenta os valores considerados para cada atividade. A partir de um modelo performabilidade (Seção 4.3), a métrica é estimada para os cenários concebidos (ver Tabela 4.4). Em tal situação, o cenário 3 tem uma melhor vazão de entrega, o que reflete a disponibilidade anteriormente calculada. Sempre que o valor não é satisfatório, o gerente poderia, então, avaliar as ações para melhorar a métrica, considerando, por exemplo, as alternativas descritas nos parágrafos anteriores.

Tabela 4.3: Atribuídos Temporizados das Atividades em Meses.

Atividade	Valor
Análise de Requisitos	2,0
Projeto	1,0
Implementação	6,0
Teste	2,0
Implantação	1,0

4.3 Modelos Propostos

Esta seção apresenta os modelos adotados com base em RBD e SPN para estimar as métricas de dependabilidade e performabilidade. Como se segue, os modelos e bloco de

Tabela 4.4: Resultados da Vazão dos Cenários em Meses.

Cenário	Vazão
1	0,8003296
2	0,9692717
3	0,9985114

construção são apresentados.

4.3.1 Modelos RBD

Este trabalho adota modelos RBDs, por serem um modelo combinatorial, é possível calcular as métricas de disponibilidade e confiabilidade por meio de fórmulas fechadas. Portanto, uma das vantagens de utilizar o RBD é a facilidade de avaliar a confiabilidade e disponibilidade dos sistemas.

Os modelos RBDs mais simples e comuns dão suporte a arranjos em séries e paralelo. Para que um sistema esteja em funcionamento é necessário que todos os componentes estejam operacionais (funcionando) os blocos destes modelos serão conectados em série. No entanto, uma conexão em série representa uma dependência direta entre os componentes. Consequentemente, um sistema pode funcionar com apenas um componente operacional, dessa forma os blocos estão conectados em paralelo. Assim, uma conexão em paralelo é utilizada para representar redundância.

Os arranjos K-out-of-n descrevem estruturas em que o sistema pode funcionar se k ou mais componentes estão no estado operacional (Kuo & Zuo 2003). Por conseguinte, uma estrutura formada por quatro componentes e necessita de dois funcionamento para prover o serviço esperado, assim temos uma estrutura 2-out-of-4 (ou 2 de 4). Os arranjos em série e paralelo são casos especiais do arranjo K-out-of-n, ou seja, um arranjo em série é uma n-out-of-n e um arranjo em paralelo é uma 1-out-of-n (Kuo & Zuo 2003).

Os arranjos em RBD adotados nesta dissertação foram descritos no Capítulo 3, tais como arranjos em Série, Paralelo e K-out-of-n.

4.3.2 Modelos SPN

Este trabalho adota Redes de Petri Estocástica (SPN) (Balbo 2001), que permite a associação de distribuição exponencial para transições temporizadas (representadas por retângulos brancos), ou zero atraso nas transições imediatas (descrito como retângulos pretos finos). O espaço de estados de modelos SPN pode ser traduzido em tempo con-

tínuo de cadeias de Markov (CTMC) (Balbo 2001), bem como técnicas de simulação podem ser adotadas para estimar as métricas de dependabilidade e performabilidade, como uma alternativa para a geração da cadeia de Markov.

4.3.2.1 Componente Simples

O componente simples é um bloco de construção representativo (Silva & et al. 2013). Ele é caracterizado pela ausência de redundância, no sentido de que o componente pode estar em dois estados: ativo ou inativo. A Figura 4.3 mostra o respectivo componente. A Tabela 4.5 descreve o significidado dos lugares e a Tabela 4.6 descreve o significado das transições do modelo de componente simples.

O componente simples possui dois parâmetros (não mostrados na figura), ou seja, X_MTTF e X_MTTR, os quais representam os tempos associados às transições $X\_Failure$ e X_Repair, respectivamente. Lugares X_ON e X_OFF representam estados ativos e inativos.

Se X_MTTF e X_MTTR são exponencialmente distribuídos, serão os únicos parâmentos necessários para o cálculo da disponibilidade.

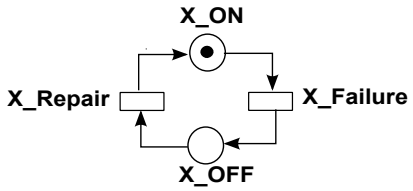

Figura 4.3: Modelo Componente Simples.

Tabela 4.5: Descrição dos Lugares do Componente Simples.

Lugar	Descrição
X_ON	Componente ativo ou desenvolvedor disponível
X_OFF	Componente inativo ou desenvolvedor indisponível

Tabela 4.6: Descrição das Transições do Componente Simples.

Transição	Descrição
X_Failure	Representa a falha do componente/desenvolvedor
X_Repair	Representa o reparo do componente/desenvolvedor

Sempre que $\#X\_ON > 0$, o componente está operacional, por conseguinte, $P\{\#X\_ON > 0\}$ indica a respectiva disponibilidade. Um modelo componente simples pode representar um único desenvolvedor em relação à rotatividade ou implementação de requisitos.

Foi analisado e verificado um conjunto de propriedades estruturais e comportamentais associadas ao modelo. As seguintes propriedades de interesse foram satisfeitas: limitada, segura e ausência de *deadlocks*.

4.3.2.2 Modelo *Cold Standby*

A Figura 4.4 representa o modelo *cold standby* (Silva & et al. 2013, Guimarães & et al. 2013), no qual um componente de reposição não ativo só é ativado quando o componente principal falha (X_OFF). Em tal caso, a transição $Activate\_Spare\_X$ representa o módulo de reposição para começar a operação. A ativação leva um certo período de tempo, o qual é representado pelo tempo médio de ativação ($MTActivate$). Depois de corrigir o componente principal ($\#X\_ON > 0$), a reposição está desativada (disparo da transição $Deactivate\_Spare\_X$). A Tabela 4.7 descreve o significado dos lugares e a Tabela 4.8 descreve o significado das transições do modelo.

Representar a redundância *cold standby* utilizando RBD não é fácil, devido à estrutura do sistema ser dinâmica, isto é, ela muda ao longo do tempo, e o tempo para ativar o mecanismo de reposição deve ser representado. Dessa forma, o modelo de *cold standby* é apresentado em SPN.

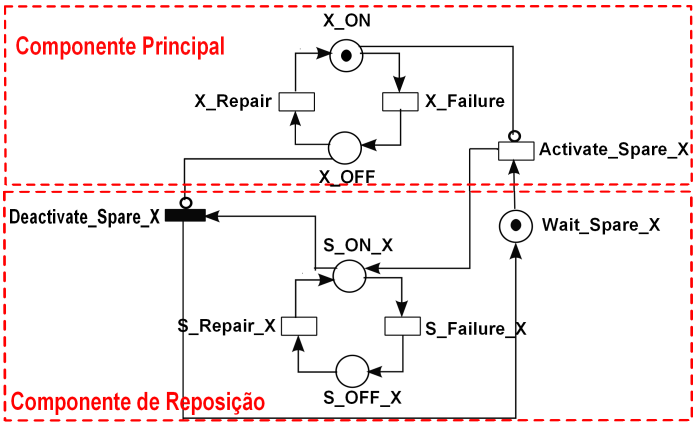

Figura 4.4: Modelo *Cold Standby*.

Tabela 4.7: Descrição dos Lugares do Modelo *Cold Standby*.

Lugar	Descrição
X_ON	Componente ativo ou desenvolvedor disponível do módulo principal
X_OFF	Componente inativo ou desenvolvedor indisponível do módulo principal
S_ON_X	Componente ativo ou desenvolvedor disponível do módulo de reposição
S_OFF_X	Componente inativo ou desenvolvedor indisponível do módulo de reposição
Wait_Spare_X	Representa a espera para ativar o módulo de reposição

Tabela 4.8: Descrição das Transiçõe do Modelo *Cold Standby*.

Transição	Descrição
X_Failure	Representa a falha do componente/desenvolvedor do módulo principal
X_Repair	Representa o reparo do componente/desenvolvedor do módulo principal
S_Failure_X	Representa a falha do componente/desenvolvedor do módulo de reposição
S_Repair_X	Representa o reparo do componente/desenvolvedor do módulo de reposição
Activate_Spare_X	Representa a ativação do módulo de reposição
Deactivate_Spare_X	Representa a desativação do módulo de reposição

Semelhante ao modelo de componente simples, X_Repair e S_Repair_X representa as atividades de manutenção, bem como $X\_Failure$ e $Failure\_S\_X$ denotam os atrasos de falha. Além disso, a expressão $P\{(\#X\_ON + \#S\_ON\_X) > 0\}$ indica o modo de funcionamento do componente, assim, $P\{\#(X\_ON + \#S\_ON\_X > 0)\}$ indica a respectiva disponibilidade.

Esse modelo pode ser adotado para representar um desenvolvedor de *backup*, a fim de diminuir o impacto em relação à rotatividade de desenvolvedor.

Foi analisado e verificado um conjunto de propriedades estruturais e comportamentais associadas ao modelo. As seguintes propriedades estruturais e comportamentais associadas ao modelo *Cold Standby* foram satisfeitas: alcançabilidade, segura e ausência de *deadlocks*.

4.3.2.3 Modelo Manutenção Preditiva/Reuniões Periódicas

Os modelos apresentados até então consideraram manutenção corretiva. Nesta seção, um modelo de manutenção preditiva é mostrado, que leva em conta as ações periódicas para manter o componente no estado operacional. No contexto deste trabalho, manutenção preditiva assume reuniões periódicas com os *stakeholders* para ajudar os desenvolvedores a evitar falhas relativas à implementação de requisitos, e uma abordagem semelhante pode ser adotada para rotatividade de desenvolvedor (ou análise de requisitos).

A Figua 4.5 descreve o modelo em que um componente simples está conectado a um bloco preditivo.

A transição *MTBP* representa o tempo médio entre manutenções preditivas (por exemplo, reuniões), e a transição *PM* considera a execução da manutenção preditiva (a reunião em si). A Tabela 4.9 descreve o significado dos lugares e a Tabela 4.12 descreve o significado das transições do modelo.

O lugar *Recurso* indica que o recurso está disponível para tal manutenção, por exemplo, um *stakeholder*, no qual as linhas tracejadas indicam que o recurso é compartilhado com outros componentes simples.

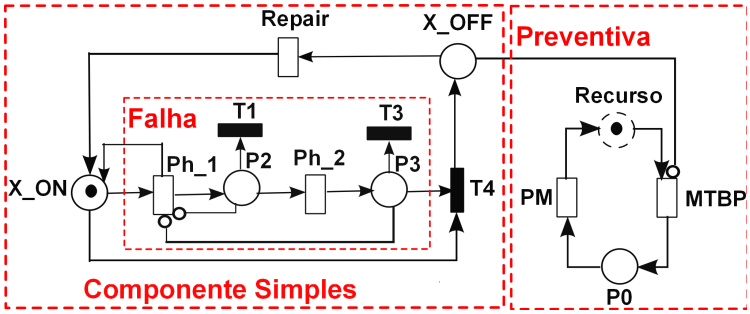

Figura 4.5: Modelo Manutenção Preditiva/Reunião Periódica.

Tabela 4.9: Descrição dos Lugares do Modelo Manutenção Preditiva.

Lugar	Descrição
X_ON	Componente ativo ou desenvolvedor disponível
X_OFF	Componente inativo ou desenvolvedor indisponível
P2 e P3	Representa as fases da Erlang
Recurso	Representa Stakeholder
P0	Recurso não disponível

Tabela 4.10: Descrição das Transições do Modelo Manutenção Preditiva.

Transição	Descrição
Ph_1 e Ph_2	Representa momentos em que o desenvolvedor estar implementado um requisito, podendo ocorrer a falha
T1, T2 e T3	Representa o reset da contagem do tempo
MTBP	Representa tempo médio entre manutenções preditivas
PM	Tempo da manutenção/reunião
Repair	Representa o reparo do componente/desenvolvedor

Como a distribuição exponencial tem a propriedade sem memória, a manutenção preditiva não tem como impacta sobre o decorrer do tempo de falha. No entanto, tempos de falha baseadas em *phase-type distributions* são afetados. Assim, como um exemplo proeminente, a Figura 4.5 retrata o MTTF assumindo uma Erlang com duas fases. Dessa forma, um *token* é mantido armazenado no lugar X_ON durante o disparo de todas as fases Erlang, e, seguidamente, é retirado quando a transição $T4$ dispara, concluindo toda a fase de disparo e completando a execução da transição Erlang. Para cada lugar que representa uma fase, uma transição imediata é adotada como transição de saída com a expressão de guarda $(\#P0 > 0)$. Assim, sempre que uma manutenção preditiva está em execução (por exemplo, uma reunião periódica), o decorrer do tempo de falha é resetado, melhorando a métrica de dependabilidade. Da mesma forma, $\#X\_ON > 0$ avalia se o componente está operacional.

4.3.3 Modelo de Performabilidade

O modelo de performabilidade é baseado em uma abordagem de modelagem hierárquica que também considera os resultados dos modelos de dependabilidade. Neste trabalho, o modelo de performabilidade representa um processo em cascata, em que as atividades de desenvolvimento de *software* estão em sequência.

A Figura 4.6 representa o modelo de performabilidade, em que um componente simples representa o grupo de desenvolvedores alocados para a atividade de implementação (transição *Impl*). A transição *Impl* é habilitada quando tem um *token* no lugar *ON*. Outros componentes simples poderiam ser alocados para outras transições, mas, neste trabalho, estamos preocupados com o impacto das falhas de desenvolvedores durante a implementação.

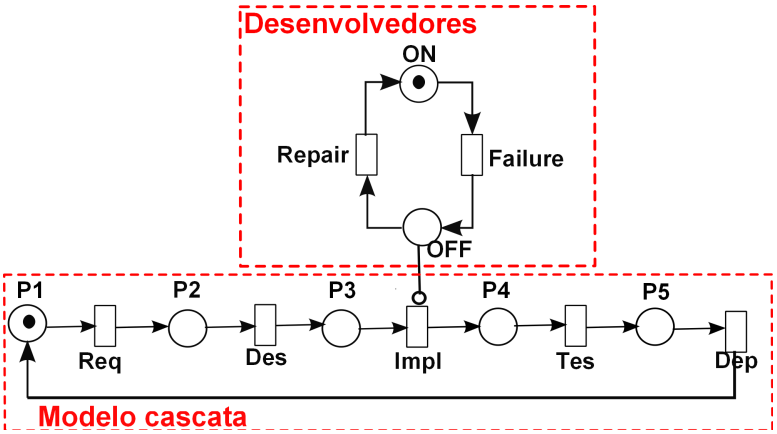

Figura 4.6: Modelo Performabilidade/Cascata

A Tabela 4.11 descreve o significado dos lugares e a Tabela 4.12 descreve o significado das transições do modelo. A vazão de entrega é a métrica de interesse, e é calculada utilizando $P\{\#P5 > 0\} \times W(Dep)$. Em outras palavras, a métrica assume o número de funcionalidades entregues por unidade de tempo.

Tabela 4.11: Descrição dos Lugares do Modelo de Performabilidade.

Lugar	Descrição
ON	Grupo de desenvolvedor ativo
OFF	Grupo de desenvolvedor inativo
P1	Início da atividade de análise de requisitos
P2	Início da atividade de projeto
P3	Início da atividade de implementação
P4	Início da atividade de teste
P5	Início da atividade de implantação

Tabela 4.12: Descrição das Transições do Modelo de Performabilidade.

Transição	Descrição
Failure	Representa a falha do grupo de desenvolvedores
Repair	Representa o reparo do grupo de desenvolvedores
Req	Representa a atividade de análise de requisitos
Des	Representa a atividade de projeto
Impl	Representa a atividade de implementação
Tes	Representa a atividade de testes
Dep	Representa a atividade de implantação

Foi analisado e verificado um conjunto de propriedades estruturais e comportamentais associadas a tal modelo. As seguintes propriedades de interesse foram satisfeitas: alcançabilidade, limitada, segura e ausência de *deadlocks*.

4.3.4 Composição Hierárquica

Como foi referido anteriormente, RBD e SPN são técnicas de modelagem importantes para avaliar a dependabilidade do sistema, mas ambos têm limitações. Assim, podemos adotar um mecanismo de modelagem híbrida, que é muito adequado para a representação de sistemas grandes e complexos. Além disso, a composição hierárquica também é levada em conta (Maciel et al. 2011).

Em geral, o sistema é decomposto em subsistemas, que consideram apenas os componentes que estão relacionados sobre a ativação, falha ou reparo. De fato, tal decomposição pode considerar subsistemas com apenas um componente, no sentido de que a ativação, a falha e reparo não dependem de qualquer outro componente do sistema. Modelos menores (RBD ou SPN) são, então, criados para cada subsistema; as respectivas métricas são estimadas; e o modelo do sistema final (RBD ou SPN) é criado utilizando tais métricas para avaliar a dependabiliddade do sistema.

A Figura 4.7 representa um exemplo, no qual 4 desenvolvedores (DE) são considerados. Três pessoas são necessárias para manter o sistema (projeto de *software*) em funcionamento e um arranjo série é adotado. Dois desenvolvedores (DE_1 e DE_2) são representados por blocos separados, pois eles não são dependentes de outros componentes, considerando as questões de confiabilidade. Por outro lado, duas pessoas estão relacionadas por programação em par baseada em *cold standby* (DEP), e um SPN é adotado no presente caso. Assim, o modelo SPN é avaliado, inicialmente e, em seguida, a estimativa da métrica (por exemplo, a disponibilidade) é considerada no modelo de RBD. Como alternativa, um único modelo SPN, não-hierárquico poderia ser adotado para todo o sistema, ao custo de gerar um espaço de estado maior ou tempo de simulação.

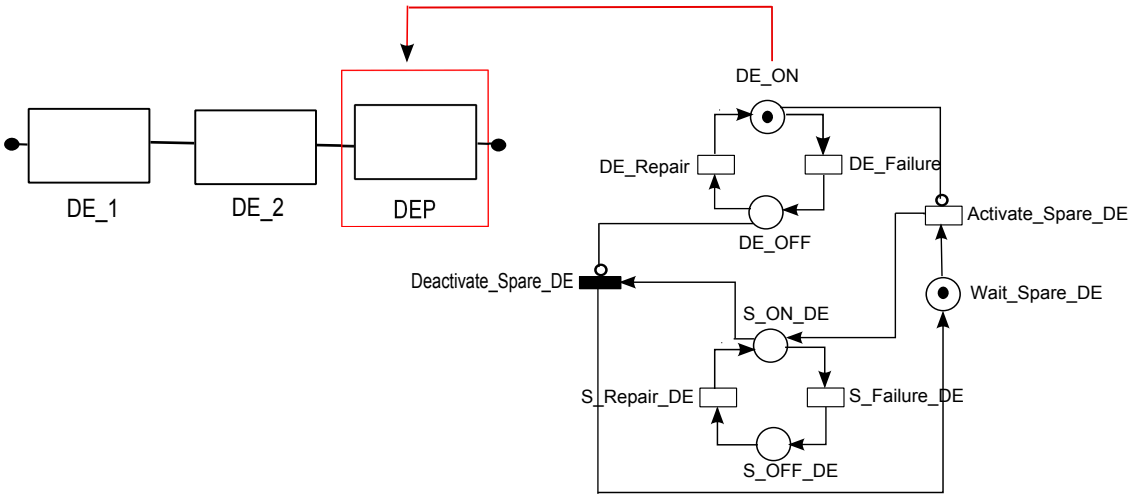

Figura 4.7: Composição Hierárquica

Adicionalmente, a modelagem hierárquica é importante para avaliar métricas de performabilidade. A partir do modelo de dependabilidade final, o MTTF e MTTR podem ser estimados, e estes valores podem ser considerados em um modelo de performabilidade, ao invés de adotar um modelo baseado no estado geral. Os estudos de caso concebidos adotam essa abordagem.

4.4 Considerações Finais

Este capítulo apresentou a metodologia proposta para avaliação de performabilidade de riscos de desenvolvimento em projetos de *software* e um exemplo motivacional. Em seguida, foram apresentados os modelos adotados com base em diagramas de blocos de confiabilidade e redes de Petri estocásticos para estimar as métricas de dependabilidade e performabilidade. Para cada modelo SPN, foram apresentadas expressões para cálculo de algumas métrica de dependabilidade e performabilidade.

5

Estudos de Caso

A única maneira de fazer um excelente
trabalho é amar o que você faz.

—STEVE JOBS

Neste capítulo serão apresentados dois estudos de casos para mostrar aplicabilidade da metodologia e dos modelos propostos. Em seguida, vários cenários serão apresentados com o objetivo de avaliar o impacto da performabilidade de riscos de desenvolvimento em projetos de *software*. Assim, todas as atividades da metodologia foram utilizadas para auxiliar na avaliação desses estudos.

Os estudos de caso propostos levam em conta dois projetos de *software* do mundo real para demonstrar a abordagem concebida para avaliar os riscos, a fim de auxiliar na tomada de decisão para gerentes de projeto. O estudo de caso 1 trata apenas de dependabilidade, e o estudo de caso 2 leva em conta a performabilidade.

Para uma melhor visualização, este trabalho adota *number of nines* (número de noves) (Maciel et al. 2011) para apresentar alguns resultados, utilizando a seguinte equação: $-log_{10}(1-X)$ (onde X refere-se à disponibilidade ou confiabibilidade). Por exemplo, $0,99521$ pode ser apresentado como $-log_{10}(1-0,99521) = 2,319664487$.

Inicialmente, os parágrafos seguintes descrevem sobre MTTF e MTTR estimados. Em seguida, um estudo de caso é apresentado considerando rotatividade de desenvolve-

dor e estudo de caso 2 assume os riscos relacionados com a implementação de requisitos.

5.1 Estimando MTTF e MTTR

Considerando-se as atividades do método proposto, os estudos de caso adotar dados históricos para estimar MTTF e MTTR.

Em relação ao MTTF, a censura está presente, e os dados são classificados como tipo I, isoladamente censurado à direita (Seção 3): a observação terminou após um determinado período de tempo, todos os desenvolvedores tiveram o mesmo período de observação, e alguns indivíduos não deixaram o projeto durante a observação. Neste caso, a Capítulo 4 explica a abordagem adotada. Por outro lado, MTTRs são estimados com base em dados completos (sem censura).

5.1.1 Estudo de Caso 1

Este estudo de caso assume rotatividade de desenvolvedor como um risco potencial. O projeto contempla 10 desenvolvedores, que podem ser *trainee* (TR), desenvolvedor *junior* (JE) ou desenvolvedor *senior* (SE). Mais especificamente, um desenvolvedor sênior tem mais experiência do que as outras categorias, e um desenvolvedor júnior é mais experiente do que os trainees.

É necessário um mínimo de 7 desenvolvedores (sem discernir categorias anteriores) para manter o projeto operacional e, com base nos dados coletados, a disponibilidade estimada é de $0,999214286$.

Nas próximas seções, diferentes cenários são avaliados, incluindo a adoção de redundância.

5.1.1.1 Cenários sem Distinção Desenvolvedor

Estes cenários adotam a abordagem k-out-of-n a fim de avaliar a disponibilidade e confiabilidade, assumindo um número mínimo de desenvolvedores (k) para não interromper o projeto de *software*. MTTF representa o tempo médio para um desenvolvedor deixar o projeto e MTTR considera o tempo médio para a substituição de um desenvolvedor. Nesta seção, os valores médios são calculados levando-se em conta todas as categorias. A Tabela 5.1 representa os valores e as distribuições de probabilidade escolhidas.

Ambos modelos combinatórios são baseados em estado e são representações adequadas. A Figura 5.1 representa o modelo RBD adotado (como uma equação de forma

fechada está disponível).

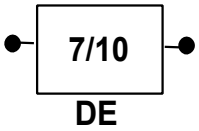

Figura 5.1: Modelo RBD sem Distinção de Desenvolvedor.

Tabela 5.1: MTTF/MTTR em Meses sem Distinção de Desenvolvedor.

Métrica	Valor	Desvio padrão	Distribuição
MTTF	23,3385	12,3405	Hipoexponencial
MTTR	2,0	1,0	Erlang

A Tabela 5.2 apresenta os resultados da disponibilidade e a Figura 5.2 mostra uma comparação em relação a número de noves. O modelo base é 7-*out*-of-10, em que a disponibilidade é $0,994472645$. O erro relativo é $0,47\%$ comparando com a disponibilidade estimada para o projeto de *software* real.

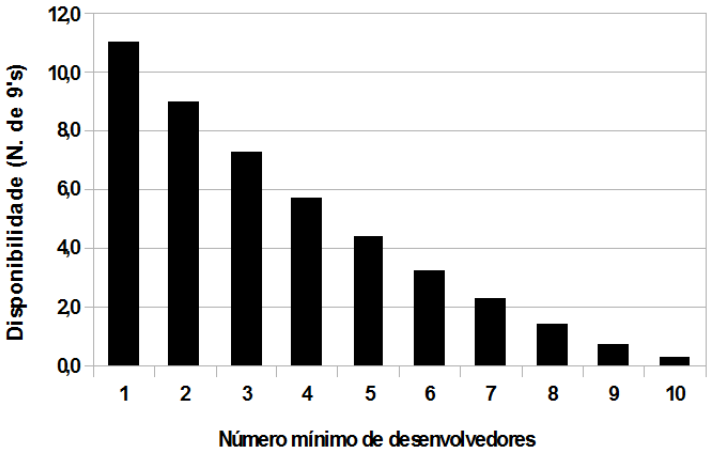

Figura 5.2: Disponibilidade para Desenvolvedor sem Distinção em *k*-*out*-of-*n*.

Como esperado, quando o número mínimo de desenvolvedores necessários aumenta, a disponibilidade diminui. Particularmente, a partir do mínimo de 4 desenvolvedores, a disponibilidade fica abaixo de um mínimo aceitável (podendo levar a falha do projeto). Assim, o gerente do projeto pode evitar cenários com baixa disponibilidade, buscando alternativas para aumentar o MTTF, atráves de incentivos aos profissionais (por exemplo, aumento de salários e gratificações adicionais por produção).

A Tabela 5.3 apresenta resultados de confiabilidade ao longo de um período de 12 meses e a Figura 5.3 mostra uma comparação em relação a número de noves. Seme-

Tabela 5.2: Resultados da Disponibilidade para Cenários em *k-out-of-n*.

k-out-of-10	Disponibilidade	Disponibilidade (N. de 9's)
1	0,999999999990613	11,0274710217
2	0,999999998895311	8,95675996780
3	0,999999941000000	7,22914798840
4	0,999998152000000	5,73329803310
5	0,999961600000000	4,41566877560
6	0,999449776000000	3,25946047020
7	0,994472645000000	2,25748264150
8	0,961284687000000	1,41211722530
9	0,816056393000000	0,73531530160
10	0,439458250000000	0,25139203500

lhante à disponibilidade, a confiabilidade diminui quando k é aumentado. Para $k > 9$, a confiabilidade é muito baixa (isto é, alta probabilidade de falha).

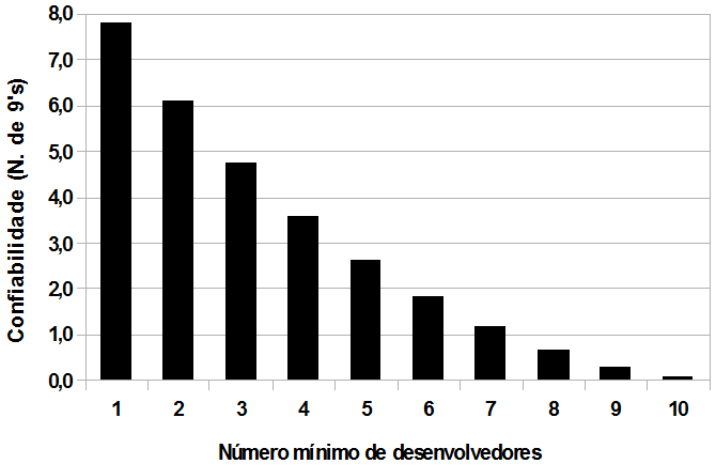

Figura 5.3: Confiabildade para Desenvolvedor sem Distinção em *k-out-of-n*.

Tabela 5.3: Resultados da Confiabilidade para Cenários em k-out-of-n.

k-out-of-10	Confiabilidade	Confiabilidade (N. de 9's)
1	0,999999984715372	7,8157451297
2	0,999999213226689	6,1041503807
3	0,999981689891544	4,7373090832
4	0,999745826687782	3,5948700516
5	0,997662425363553	2,6312345136
6	0,985043311014274	1,8251645371
7	0,931964360959178	1,1672635316
8	0,778870079508063	0,6553524902
9	0,489092400007291	0,2916576369
10	0,164059658900933	0,0778247159

5.1.1.2 Cenários com Desenvolvedores Distintos

Este experimento considera o impacto de categorias de desenvolvedores em dependabilidade. A Tabela 5.4 representa os MTTFs estimados e as distribuições de probabilidade para cada categoria. Além disso, assumimos o mesmo MTTR para todos os tipos de especialização (Tabela 5.1), e a equipe contempla três *trainees* (TR), 2 desenvolvedores *junior* (JE) e cinco desenvolvedores *senior* (SE).

Tabela 5.4: MTTF em Meses Distintos para Desenvolvedores.

Especialização	MTTF	Desvio padrão	Distribuição
TR	20,5676	10,5147	Hipoexponencial
JE	23,1059	22,1457	Hipoexponencial
SE	31,8521	17,2144	Hipoexponencial

A Tabela 5.5 apresenta alguns cenários concebidos, requerendo um mínimo de 7 desenvolvedores, mas com diferentes restrições para cada categoria. Todos os cenários têm 10 desenvolvedores, mas, por exemplo, o cenário 1 requer pelo menos um *trainee*, um desenvolvedor *junior* e 5 desenvolvedores *senior* para estar operacional. Ambos os modelos combinatórios e baseados em estado são abordagens viáveis para modelar os cenários e a Figura 5.4 representa o modelo RBD para o cenário 1. Outros cenários adotam modelos semelhantes, mas com um diferente valor *k* para cada tipo de especialização.

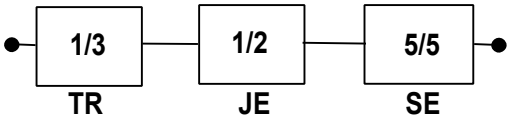

Figura 5.4: Modelo RBD para Cenário 1.

Tabela 5.5: Número Mínimo de Desenvolvedores para Cada Cenário.

Cenário	TR	JE	SE
1	1	1	5
2	3	1	3
3	2	2	3
4	3	2	2

A Tabela 5.6 apresenta os resultados, e a Figura 5.5 retrata resultado da disponibilidade considerando número de noves. O Cenário 3 tem o melhor valor de disponibilidade, mas a disponibilidade (A) é menor do que os valores apresentados na Tabela 5.2.

A partir da indisponibilidade ($UA = 1 - A$), onde A é a disponibilidade, um indivíduo pode estimar possíveis custos relacionados com a paralisação do projeto. Por exemplo,

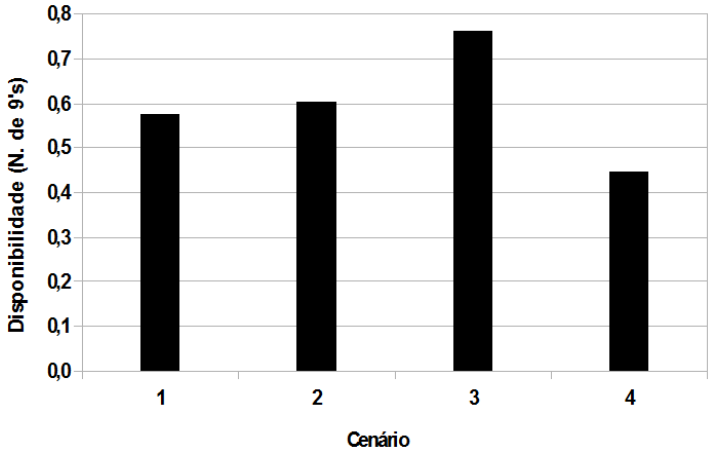

Figura 5.5: Disponibilidade para Desenvolvedores Distintos em *k*-out-of-*n*.

Tabela 5.6: Resultados para Desenvolvedores Distintos.

Cenário	Disponibilidade	Disponibilidade (N. de 9's)
1	0,732310089	0,572367997
2	0,750777023	0,603411921
3	0,826682487	0,761157551
4	0,641155997	0,445094307

supondo que uma multa mensal (MF) de US$ 50.000,00 é aplicada, devido à paralisação, o respectivo custo (C) pode ser calculado utilizando C = MF $\times$ UA $\times$ P, em que P é o período. A Tabela 5.7 mostra as penalidades para cada cenário tendo em conta períodos diferentes.

Tabela 5.7: Penalidades em US$.

Cenário	1 Mês	2 Mês	3 Mês
1	13.384,50	26.768,99	40.153,49
2	12.461,15	24.922,30	37.383,45
3	8.665,88	17.331,75	25.997,63
4	17.942,20	35.884,40	53.826,60

5.1.1.3 Programação em Pares e Desenvolvedor de *Backup*

É um dos aspectos mais comentados da metodologia *(XP)* (Pressman 2006). A implementação é feita em dupla, ou seja, dois desenvolvedores trabalham em um único computador. Eles irão trabalhar em conjunto para apresentar uma melhor solução para a implementação de uma funcionalidade do *software* em desenvolvimento. Na prática, cada desenvolvedor assume papel ligeiramente diferente, no qual, são comumente cha-

mados de piloto e copiloto (Pressman 2006). Por exemplo, um desenvolvedor poderia ser responsável por comandar o computador, digitar o código, enquanto o outro será auxiliar e revisará o código, garantindo que as normas de codificação estão sendo seguidas. Consequentemente, os papéis serão trocados durante o desenvolvimento do *software*.

O projeto de *software* poderia adotar a programação em pares e desenvolvedor de *backup* para os desenvolvedores *senior* (SE), como eles têm mais experiência, são normalmente associados com as atividades mais importantes na construção de *software*. De fato, quando os desenvolvedores experientes abandonam um projeto de *software*, eles criam uma lacuna de conhecimento que tem de ser controlada (Izquierdo-cortazar & et al. 2010). Duas abordagens são assumidas: *cold* e *hot*. Este último assume que um desenvolvedor adicional está ativo, e o anterior considera um atraso para ativar o indivíduo de reposição.

Modelos combinatórios não são muito adequados para representar o comportamento dinâmico de redundância *cold standby*, mas os modelos baseados em estado são viáveis. Para reduzir o tamanho do espaço de estado, cada SE com *cold standby* é representado por um modelo SPN (Capítulo 4) e o modelo final é um RBD. A Figura 5.6 representa o modelo, no qual as distribuições do tipo fase (Erlang e Hipoexponencial) são explicitamente representadas por falha e reparo (Watson & A. 1991). O tempo médio para ativar é um dia. A partir do modelo *cold standby*, a disponibilidade é estimada $(P\{(\#X\_ON + \#S\_ON\_X) > 0\})$ para 1 SE, e o respectivo valor é adotado no modelo RBD (por exemplo, Figura 5.4.) para estimar a disponibilidade de um cenário. No que diz respeito ao *hot standby*, dois desenvolvedores *senior* estão em arranjo paralelo. A respectiva disponibilidade é estimada e adotada na Figura 5.7 (em vez de considerar a disponibilidade do *cold standby*).

A Tabela 5.8 apresenta os resultados e a Figura 5.8 os descreve, indicando que o cenário 1 é melhorado consideravelmente, uma vez que requer um mínimo de 5 SEs para manter o projeto operacional. Os outros cenários não são significativamente afetados, uma vez que requer menos SEs, e TRs, bem como JEs não tem reposição de desenvolvedores. No entanto, a técnica *hot standby* proporciona maior disponibilidade do que a contrapartida da *cold* para um cenário.

Com os experimentos propostos, foi possível avaliar o risco proeminente em diferentes configurações. Ademais, as técnicas baseadas em redundância dinâmica são abordagens possíveis para melhorar as métricas de dependabilidade.

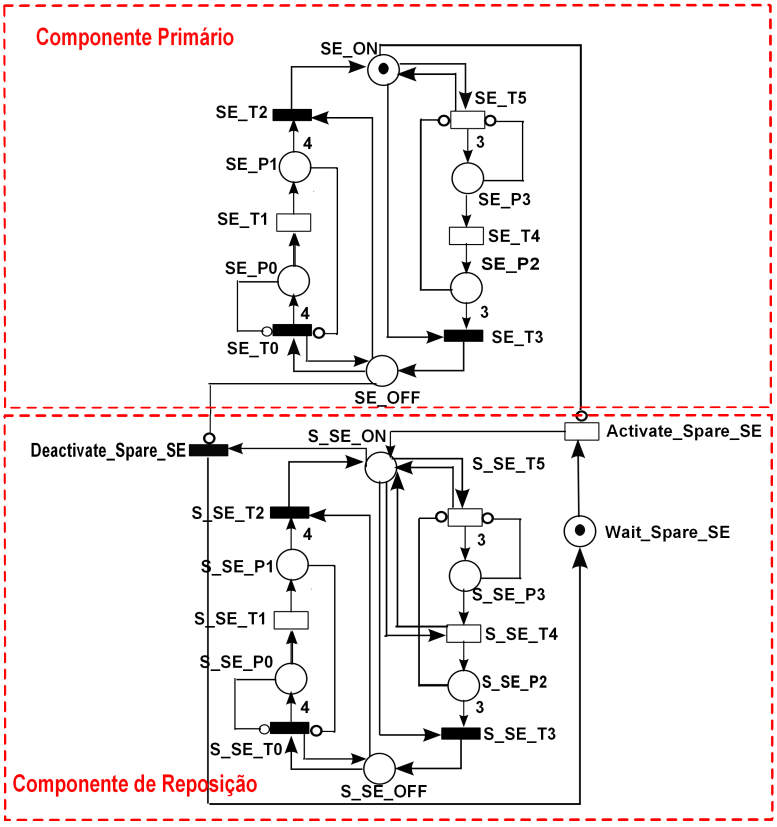

Figura 5.6: Modelo SPN *cold standby*.

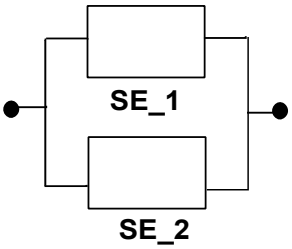

Figura 5.7: Modelo RBD *hot standby*.

Tabela 5.8: Resultados para *Cold/Hot standby*.

Cenário	Dis. Cold	Dis. (N. de 9's) Cold	Dis. Hot	Dis. (N. de 9's) Hot
1	0,922313764	1,109655920	0,992962256	2,1525665348
2	0,752170840	0,605847595	0,752193670	0,6058876041
3	0,828217221	0,765020376	0,828242359	0,7650839333
4	0,641193065	0,445139172	0,641193211	0,4451393482

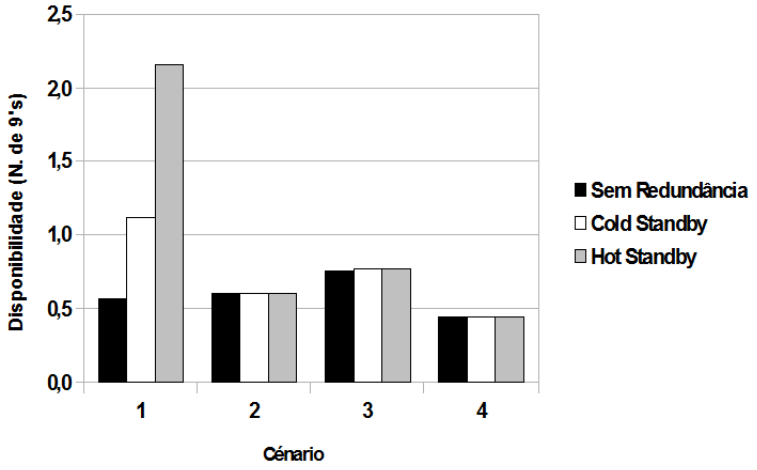

Figura 5.8: Disponibilidade para *Cold/Hot Standby*.

5.1.2 Estudo de Caso 2

Nesta seção, adota-se o método proposto para quantificar a disponibilidade e confiabilidade de um projeto de *software* sobre os riscos relacionados com a implementação de requisitos. O projeto contempla 4 desenvolvedores, e uma falha é assumida sempre que 2 desenvolvedores falham simultaneamente para implementar um requisito/funcionalidade.

A partir de dados históricos, a disponibilidade é avaliada como $0,999994419$, considerando o tempo de *uptime* e *downtime*, o que indica um erro relativo igual a $0,005\%$. Consequentemente, a disponibilidade em números de nove é $5,2532923928$.

As seções a seguir apresentam os cenários avaliados neste estudo de caso.

5.1.2.1 Mínimo de k Desenvolvedores

Um número mínimo de desenvolvedores (k) é necessário para não interromper a execução do projeto. RBD é adotada devido as equações de forma fechada.

Tabela 5.9: Métricas em Dias para o Cenário Base.

Métrica	Valor	Distribuição
MTTF	29,6298	Exponencial
MTTR	1,6923	Exponencial

A Tabela 5.10 apresenta os resultados e a Figura 5.9 mostra uma comparação em relação ao número de noves.

Tabela 5.10: Resultados de Cenários k-out-of-n para Implementação de Requisitos.

k	Disponibilidade	Disponibilidade (N. de 9's)
1	0,999991479	5,0695094347
2	0,999394687	3,2180199989
3	0,983721279	1,7883797201
4	0,800775849	0,7006580153

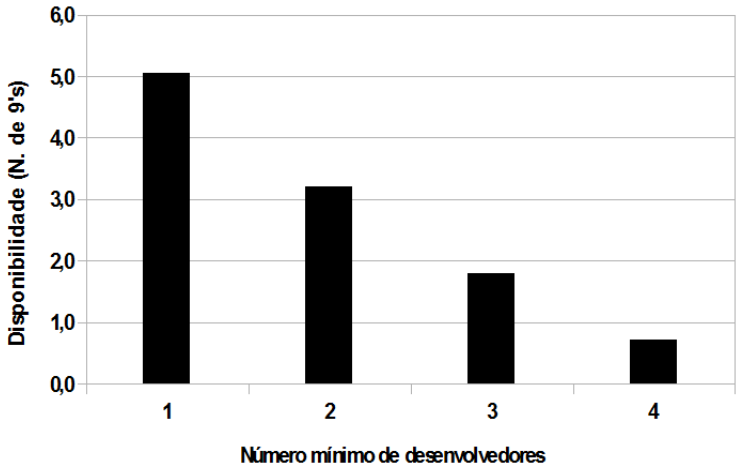

Figura 5.9: Disponibilidade para Implementação de Requisitos em *k*-out-of-*n*.

A Tabela 5.11 mostra os resultados da confiabilidade ao longo de um período de 30 dias e a Figura 5.10 apresenta uma comparação em relação a número de noves. Como se percebe, a confiabilidade do projeto em 1 mês é muito baixa e para $k > 3$ existe grande risco da falha do projeto. Ademais, semelhante à disponibilidade, a confiabilidade decresce quando *k* é acrescido num período de 1 mês.

5.1.2.2 Reuniões periódicas

São reuniões feitas com a equipe de desenvolvimento e *stakeholders* em um tempo pré-estabelecidos, visando à identificação e correção de quaisquer problemas e/ou impedimentos no processo de desenvolvimento de *software* (Schwaber & Beedle 2002). As

Tabela 5.11: Resultados da Confiabilidade para Cenários em k-out-of-n.

k-out-of-4	Confiabilidade	Confiabilidade (N. de 9's)
1	0,835673533	0,7842924819
2	0,460596911	0,2680865708
3	0,139553514	0,0652761347
4	0,017422780	0,0076333089

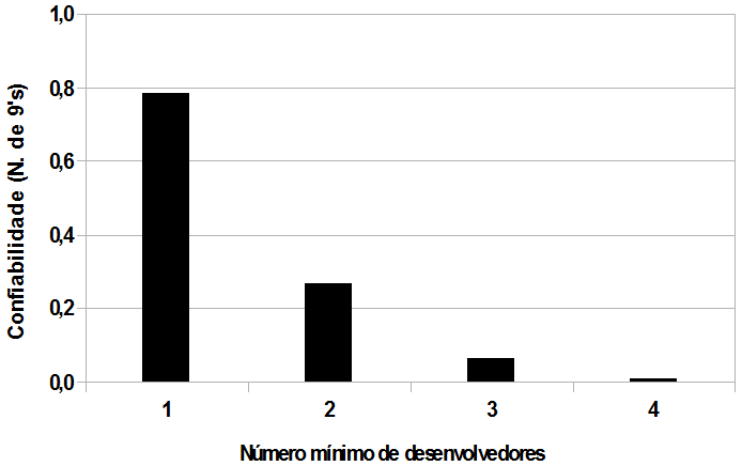

Figura 5.10: Confiabildade para Implementação de Requisitos em *k-out-of-n*.

reuniões periódicas permitem acompanhar o dia a dia de trabalho da equipe e perceber as dificuldades existentes para a realização das atividades planejadas (por exemplo, implementação de requisito/funcionalidade). O envolvimento dos *stakeholders* é importante durante as reuniões, pois sua opinião é valiosa, ou seja, entendem do negócio (*software* em construção). Dessa forma, as reuniões periódicas buscam levantar impedimentos (por exemplo, riscos) e, consequentemente, minimizá-los, com o objetivo de garantir o prazo e o custos definidos para o projeto de *software*.

A Figura. 5.11 representa o modelo adotado para reuniões periódicas utilizando SPN (Capítulo 4) como modelos combinatórios não são muito adequados para este contexto. Este modelo contempla quatro desenvolvedores que compartilham as partes interessadas, e a Tabela 5.12 representa os valores de PM, bem como MTBP em dias. MTTF e MTTR os mesmos valores anteriormente adotados, mas assumindo uma Erlang com 2 fases. Os cenários pressupõem um mínimo de k desenvolvedores, e, considerando *k-out-of-4*, a disponibilidade é definida como $P\{(\#DE\_ON\_1 + \#DE\_ON\_2 + \#DE\_ON\_3 + \#DE\_ON\_4) \geq k\}$.

A Tabela 5.13 apresenta os resultados da disponibilidade, considerando um e dois *stakeholders* (*#Stakeholder = 2*), e a Figura 5.12 descreve a comparação em relação ao número de noves. A disponibilidade em número de noves é o dobro em comparação

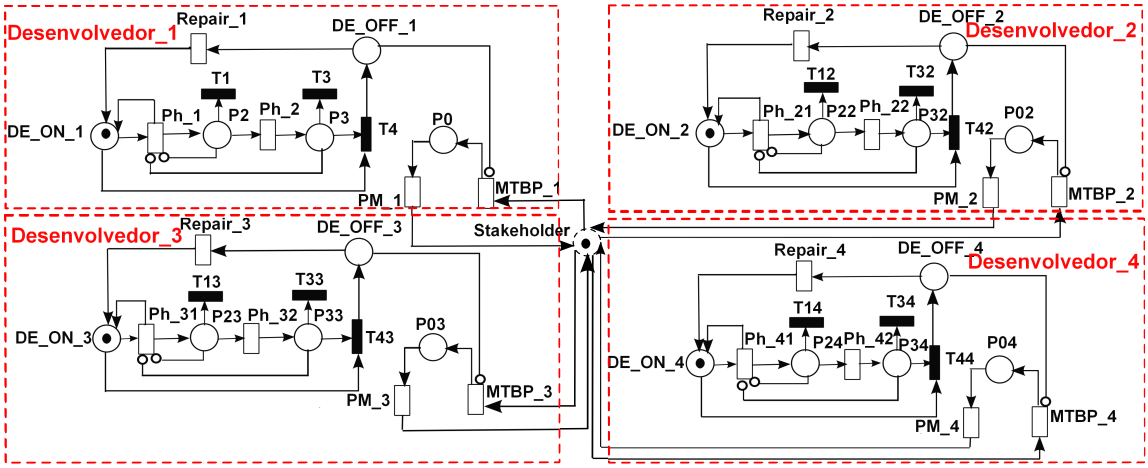

Figura 5.11: Reuniões Periódicas.

Tabela 5.12: Valores em Dias de PM e MTBP.

Transição	Valor
PM	0,0416
MTBP	1,0

com o experimento anterior, mas uma das partes interessadas adicional não melhora consideravelmente os resultados. O número de noves é duplicado em comparação com o experimento anterior, mas um *stakeholder* adicional não melhorar consideravelmente os resultados.

Tabela 5.13: Resultados da Disponibilidade Assumindo Reuniões Periódicas.

Cenário	Dis. 1 stakeh.	Dis. (N. de 9's) 1 stakeh.	Dis. 2 stakeh.	Dis. (N. de 9's) 2 stakeh.
1	0,9999999999886	10,9430954511	0,9999999999909	11,04096132700
2	0,9999999736853	7,57980157720	0,9999999790641	7,679108364500
3	0,9999781391957	4,66033386360	0,9999818724534	4,741660969800
4	0,9922359189711	2,10990994080	0,9930417736000	2,157501444100

Para o cálculo da confiabilidade, é necessário adicionar uma expressão para o peso do arco $IF((\#DE\_ON\_1 + \#DE\_ON\_2 + \#DE\_ON\_3 + \#DE\_ON\_4) = 4)$: 1 *ELSE* 2 no arco entre as transições Repair e DE_OFF em todos os desenvolvedores. Dessa forma, a transição *Repair* será inibida quando a condição de falha acontecer.

A Tabela 5.14 mostra os resultados da confiabilidade num período de 30 dias (1 mês) considerando um e dois *stakeholders*, e a Figura 5.13 apresenta a comparação em relação ao número de noves. Um *stakeholder* adicional não melhora consideravelmente os resultados da confiabilidade. No entanto, para $k > 3$, a confiabilidade é muito baixa, ou seja, probabilidade de falha do projeto.

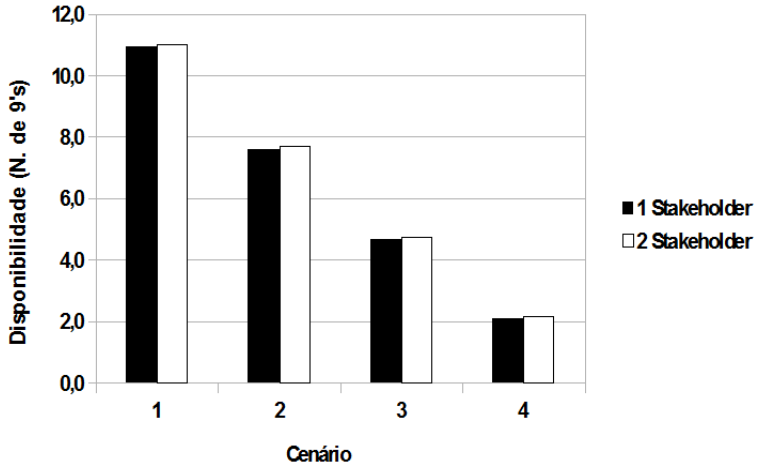

Figura 5.12: Disponibilidade com Reuniões Periódicas.

Tabela 5.14: Resultados da Confiabilidade Assumindo Reuniões Periódicas.

Cenário	Con. 1 stakeh.	Con. (N. de 9's) 1 stakeh.	Con. 2 stakeh.	Con. (N. de 9's) 2 stakeh.
1	0,999999073	6,032920265	0,999999251	6,125518182
2	0,999876215	3,907331979	0,999899970	3,999869731
3	0,994021152	2,223382487	0,994976632	2,299005005
4	0,874888764	0,902703685	0,886824530	0,946247693

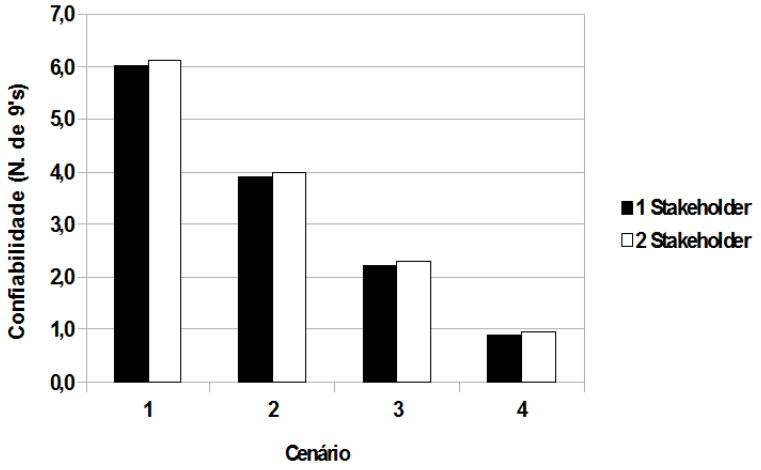

Figura 5.13: Confiabilidade com Reuniões Periódicas.

Considerando um *stakeholder*, a Figura 5.14 descreve os resultados da disponibilidade variando o tempo médio entre reuniões periódicas (de 1 a 30 dias). A disponibilidade do projeto é impactada, mas reuniões periódicas com os *stakeholders* proporcionam melhores resultados do que a abordagem sem as reuniões.

Da mesma forma que foi feito para a disponibilidade, a confiabilidade foi calculada para 1 mês, considerando um *stakeholder* e variando o tempo médio entre reuniões pe-

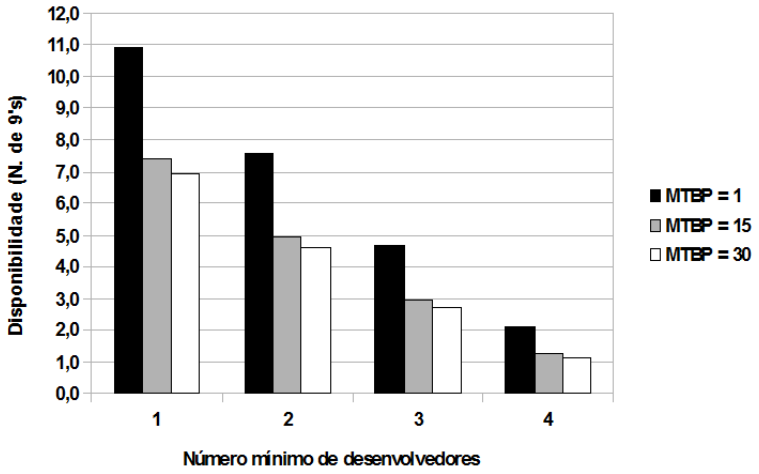

Figura 5.14: Disponibilidade para Reuniões Periódicas com Diferentes MTBPs.

riódicas (de 1 a 30 dias). A Figura 5.15 apresenta os resultados.

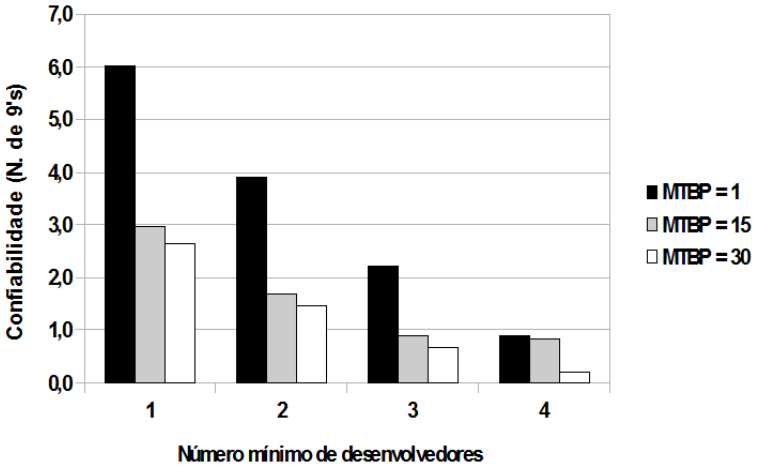

Figura 5.15: Confiabilidade para Reuniões Periódicas com Diferentes MTBPs.

5.1.2.3 Avaliação de Performabilidade

O estudo de caso assume um processo em cascata, e a métrica adotada é vazão da entrega de funcionalidades. Particularmente, esta seção assume o risco que só afeta a atividade de implementação e o modelo descrito na Figura 5.16 é adotado. A Tabela 5.15 apresenta os valores para cada transição temporizada (dias), assumindo a distribuição exponencial. Em relação aos valores das transições de falha e de reparo, a função de confiabilidade é estimada para cada cenário. O MTTF é então calculado, e MTTR é obtida por meio da equação da disponibilidade estacionária. A Figura 5.17 mostra resultados da vazão

(ano^{-1}), sem o uso de reuniões periódicas. Nota-se que, para um mínimo de 4 desenvolvedores, a vazão cai consideravelmente.

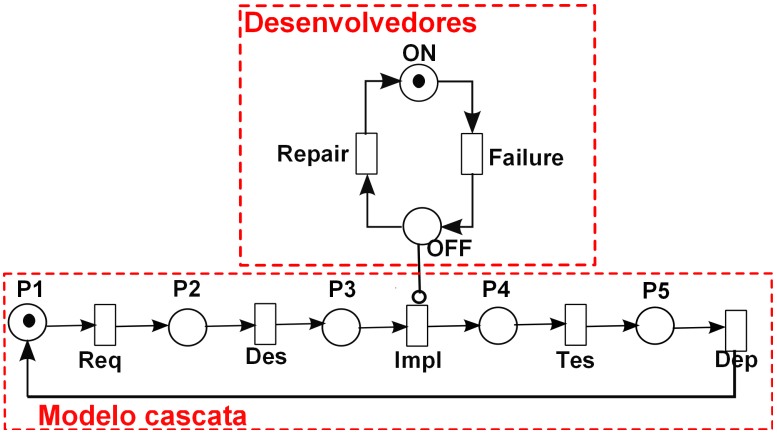

Figura 5.16: Modelo Performabilidade

Tabela 5.15: Atributos das Transição Temporizada em Dias.

Transição	Valor
Req	0,4068843001
Des	2,7281774725
Impl	9,2811921332
Tes	5,0189547346
Dep	0,5487583333

Da mesma forma, a vazão de entrega foi estimada para os cenários que adotam reuniões periódicas, e a Figura 5.18 apresenta os resultados (no ano em prol da legibilidade). Em tal situação, o aumento do tempo médio entre as reuniões periódicas (MTBP) não impacta a vazão do cenário 1 para o cenário 3, embora as reuniões periódicas melhorem a métrica de desempenho. Assim, o *stakeholder* pode ter mais flexibilidade na definição da agenda das reuniões. No entanto, para o cenário 4, o *stakeholder* tem um papel de destaque na mitigação dos riscos. As reuniões periódicas melhoram a entrega de funcionalidades, e reduzindo o MTBP o desempenho é impactado significativamente.

Os experimentos propostos neste estudo de caso demonstram que os gerentes de projetos têm uma ferramenta importante para avaliar os riscos e diferentes configurações para evitar ou mitigar esses problemas indesejáveis. Dependendo do projeto de *software* e os recursos disponíveis, técnicas baseadas na política de manutenção preditiva são abordagens possíveis para melhorar as métricas de dependabilidade, e consequentemente, pode-se realizar avaliação de performablidade do projeto de desenvolvimento de *software*.

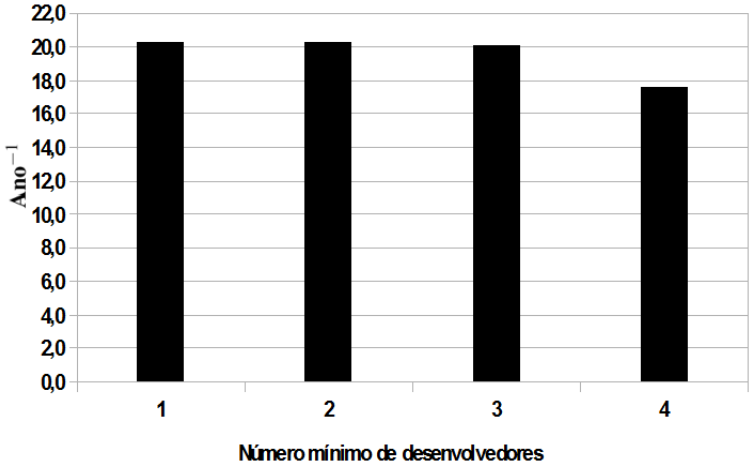

Figura 5.17: Vazão em k-out-of-n para Implemetação de Requisitos em Ano^{-1}.

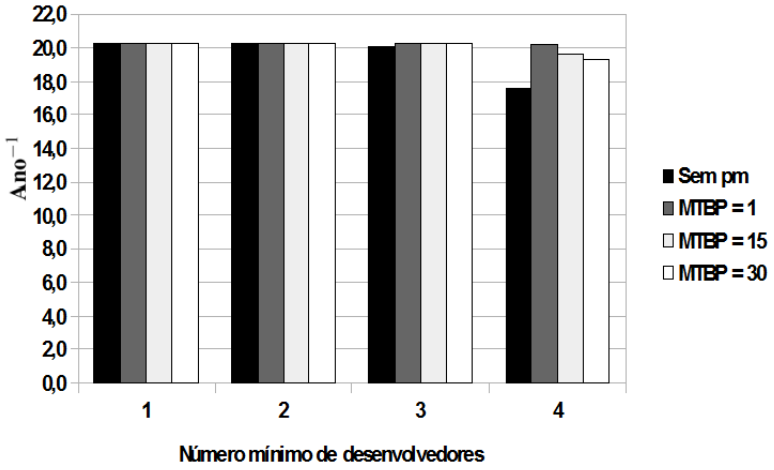

Figura 5.18: Vazão para Reuniões Periódicas com Diferentes MTBPs em Ano^{-1}.

5.2 Considerações Finais

Este capítulo apresentou os resultados obtidos na realização dos estudos de casos. Através dos estudos aqui apresentados, foi possível validar os modelos adotados como também validar a metodologia proposta. Com os resultados dos experimentos, foi possível também avaliar os projetos de *software* em termos de performabilidade.

O primeiro estudo de caso concebe a rotatividade de desenvolvedor como um risco potencial. No entanto, o projeto é composto por 10 desenvolvedores, os quais podem ser um *trainee* (TR), desenvolvedor *junior* (JE) ou um desenvolvedor *senior* (SE). A partir dessa formação da equipe, foi obtido o impacto do risco potencial sobre o projeto de desenvolvimento de *software* e, consequentemente, foram estimados os custos relacio-

nados com a indisponibilidade do projeto. Assim, foi possível mostrar os resultados das métricas de dependabilidade (disponibilidade e confiabilidade) e tais métricas também foram avaliadas utilizando técnicas de redundância dinâmicas (*Hot/Cold Stadby*).

O segundo estudo de caso descreve a implementação de requisitos como um risco potencial. Assim, o projeto é contemplado com 4 desenvolvedores. Para tal estudo, foi estimado o impacto do risco potencial sobre o projeto de desenvolvimento de *software* e, por conseguinte, foram estimadas as métricas de dependabilidade e políticas de manutenção foram adotadas para calcular tais métricas. Por fim, a métrica de performabilidade vazão de entrega foi estimada.

6

Conclusão

Você nunca sabe que resultados virão da sua ação.
Mas se você não fizer nada, não existirão resultados.

—MAHATMA GANDHI

O gerenciamento de riscos é uma parte essencial no projeto de desenvolvimento de *software*, e que desempenha um papel significativo na obtenção de um bom negócio e resultado do projeto.

O gerenciamento de riscos em projetos de desenvolvimento de *software* conta com uma atividade muito importante que é a avaliação quantitativa de riscos, pois esta envolve a definição de quais riscos são prioritários, permitindo medir as probabilidades e também estimar seus impactos para o projeto.

Muitas falhas associadas com o desenvolvimento de *software* ocorrem devido ao não tratamento dos riscos envolvidos e à fraca gestão dos mesmos. Assim, a gestão de risco eficaz tornou-se um fator essencial para assegurar o sucesso dos projetos de *software*.

Ainda hoje, poucas empresas de *software* não lidam eficientemente com riscos em seus projetos. Entretanto, gradativamente este cenário começa a ser modificado já que esse gerenciamento é apontado como a primeira dentre as atividades da gerência de projetos de *software*. A adoção de metodologias e técnicas para avaliar os riscos nesses tipos de projetos é um requisito essencial para a gerência de tais projetos. O sucesso, portanto, depende da forma com que os riscos são gerenciados durante todo o processo.

Outra notável premissa a ser considerada é que se devem estimar probabilisticamente os riscos durante a atividade de análise do processo de gerenciamento. No entanto, muitas vezes, por ser antes da execução do projeto, as estimativas podem ser realizadas através de dados históricos de projetos passados ou por experiências dos gestores e isso é um diferencial no exigente mercado competitivo, já que uma avaliação adequada dos risco é uma atribuição bem pouco empregada pelas empresas de *software*.

Este trabalho apresentou uma abordagem baseada em modelos de dependabilidade e performabilidade para avaliar os riscos de desenvolvimento de projetos de *software*. Estudos de caso do mundo real demonstraram a viabilidade da abordagem proposta, em que vários cenários foram avaliados, incluindo a adoção de redundância dinâmica e políticas de manutenção para melhorar a dependabilidade e performabilidade. Assim, este trabalho mostrou a aplicação dessas técnicas para a rotatividade de membros da equipe e implementação de requisitos, e que outros tipos de riscos podem ser avaliados.

6.1 Contribuições

As contribuições deste trabalho são as seguintes:

- Proposição de modelos SPNs e RBDs para avaliação probabilística de risco em projetos de desenvolvimento de *software*, durante a atividade de análise de risco. Através desses modelos, foi possível aferir métricas de performabilidade tais como, disponibilidade, confiabilidade e vazão em projetos de desenvolvimento de *software*;

- Desenvolvimento de uma metodologia para auxiliar os gerentes de projetos de *software* a realizarem avaliação de performabilidade dos riscos de desenvolvimento. Essa metodologia é composta por uma série de atividades, desde o modo de falha/-funcionamento e definições de métricas até a avaliação de cenários.

Além da contribuição mencionada, um artigo que apresenta alguns resultados desta dissertação foi produzido:

1. *A. Melo, E. Tavares, M. Marinho, E. Sousa, B. Nogueira and P. Maciel "Development Risk Assessment in Software Projects using Dependability Models ,"in The 2013 Thirteenth IEEE International Conference on Computer and Information Technology (CIT 2013), Sydney, Australia, 2013.*

6.2 Trabalhos Futuros

Outros estudos podem ser produzidos através dos modelos propostos e da metodologia de avaliação propostas: Em seguida, alguns deles são apresentados:

- Realizar novos experimentos, considerando outros tipos de riscos de desenvolvimento em projetos de *software*;
- Analisar outras políticas de manutenção preditiva;
- Realizar avaliação de outras métricas de desempenho (por exemplo, utilização e tempo de serviço);
- Criar novos modelos em SPN para avaliação de performabilidade em outras etapas do processo de desenvolvimento de *software* (por exemplo, análise de requisitos).

Referências

A. Schmietendorf, A. S. & Rautenstrauch, C. (2000), 'Evaluating the performance engineering process', *Proceedings of the 2nd international workshop on Software and performance* pp. 89 – 95.

Al-Rousan, T. ; Sulaiman, S. . S. R. (2009), 'Project management using risk identification architecture pattern (riap) model: A case study on a web-based application', *Asia-Pacific Software Engineering Conference (APSEC)* .

Alves, G. (2007), 'Avaliação de desempenho de cadeias de suprimentos utilizando componentes gspn'.

Amrit, T. & Mark, K. (2004), 'The one-minute risk assessment tool', *Communications of the ACM - Bioinformatics* **47 Issue 11**, 73 – 77.

Araújo, C. (2009), 'Avaliação e modelagem de desempenho para planejamento de capacidade do sistema de transferência eletrônica de fundos utilizando tráfego em rajada'.

Araújo, C. & et al. (2011), 'Performability modeling of electronic funds transfer systems', *Springer-Verlag* .

Avizienis, A., L. J. R. B. e. a. (2001), *Fundamental concepts of dependability*, Technical Report Series-University Of Newcastle Upon Tyne Computing Science.

Bakker, K., Boonstra, A. & Wortmann, H. (2009), 'Does risk management contribute to it project success? a meta-analysis of empirical evidence', *International Journal of Project Management* pp. 493 – 503.

Balbo, G. (2001), *Introduction to stochastic petri nets. Lectures on Formal Methods and Performance Analysis*.

Basit, S. & Abdullah, S. (2009), 'Risk identification, mitigation and avoidance model for handling software risk', *Hawaii International Conference on System Sciences* pp. 1 – 10.

Bolch, G., Greiner, S., de Meer, H. & Trivedi, K. S. (2006), *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, John Wiley & Sons.

Boness, K. & et al. (2008), 'A lightweight technique for assessing risks in requirements analysis', *Software, IET* **2**.

Callou, G. & et al. (2012), 'A petri net-basead approach to the quantification of data center dependability', *Computer and Information Science* .

Cassandras, C. G. & Lafortune, S. (2008), *Introduction to Discrete Event Systems*, Springer Science + Business Media, LLC.

Duarte, C.B. ; Faria, J. & Raza, M. (2012), 'Psp pair: Automated personal software process performance analysis and improvement recommendation', *Quality of Information and Communications Technology (QUATIC)* .

Ebeling, C. E. (2005), *An Introduction to Reliability and Maintainability Engineering*, Waveland Press, Inc.

Eric Bauer, R. A. & Eustace, D. (2011), *Beyond Redundancy: How Geographic Redundancy Can Improve Service Availability and Reliability of Computer-Based Systems*, Wiley-IEEE Press.

Esteves, C. & Sousa, C. (2007), *Apontamentos de ADPE.*

Falahah (2011), 'Risk management assessment using serim method', *ICEEE* .

Foo, S.-W. & Muruganantham, A. (2000), 'A software risk assessment model', *Management of Innovation and Technology (ICMIT)* .

Goel, M. K., Khanna, P. & Kishore, J. (2010), 'Understanding survival analysis: Kaplan-meier estimate', *International Journal of Ayurveda Research* **1 (4)**.

Guide, P. (2013), *A Guide to the Project Management Body of Knowledge - PMBOK Guide*, Fifth Edition.

Guide, S. G. I. (2012), Software: Global industry guide, Technical report.

Guimarães, A. & et al. (2013), 'An analytical modeling framework to evaluate converged networks through business-oriented metrics', *Reliability Engineering and System Safety* .

Gupta, D. & Sadiq, M. (2008), 'Software risk assessment and estimation model', *ICCSIT* pp. 963 – 967.

Hermanns, H., Herzog, U. & Katoen, J. (2002), 'Process algebra for performance evaluation', *Theoretical Computer Science* pp. 43 – 87.

Hu, Y. & et al. (2012), 'Software project risk analysis using bayesian networks with causality constraints', *Decision Support Systems* .

Izquierdo-cortazar, D. & et al. (2010), 'Using software archaeology to measure knowledge loss in software projects due to developer turnover *', *Second International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN)* pp. 1 – 196.

Jain, R. (1991), *The art of computer systems performance analysis*, John Wiley Sons New York.

Jawad, F. A. & Johnsen, E. (1995), 'Performability: the vital evaluation method for degradable systems and its most commonly used modelling method, markov reward modelling', *Essential Evaluation Etiquette* .

Knob, F. & et al. (2006), 'Riskfree - uma ferramenta de gerenciamento de riscos baseada no pmbok e aderente ao cmmi', *V Simpósio Brasileiro de Qualidade de Software (SBQS)* .

Kuo, W. & Zuo, M. J. (2003), *Optimal Reliability Modeling - Principles and Applications*, Wiley.

Lilja, D. J. (2000), *Measuring Computer Performance: A Practitioner's Guide*, Cambridge University Press.

Maciel, P., Lins, R. & Cunha, P. (1996), 'Introduction of the petri net and applied', *X Escola de Computação, Campinas, SP* .

Maciel, P., Trivedi, K. S., Matias, R. & Kim, D. S. (2011), *Dependability Modeling In: Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, Ed. Hershey: IGI Global, Pennsylvania, USA.

Marinho, M. (2010), 'Avaliação de desempenho de processos de testes de software'.

Marinho, M. & et al. (2010), 'Performance evaluation of test process based on stochastic models', *DEVS Integrative M&S Symposium (DEVS)* .

Marsan, M., Balbo, G., Conte, G., Donatelli, S. & Franceschinis, G. (1998), 'Modelling with generalized stochastic petri nets', *ACM SIGMETRICS Performance Evaluation Review, 26(2)* .

Meyer, J. F. (1992), 'Performability: a retrospective and some pointers to the future *', *Elsevier Science Publishers B.V.* pp. 139 – 156.

Murata, T. (1989), 'Petri nets: Properties, analysis and applications', *Proceedings of the IEEE* pp. 541 – 580.

Mustafa, N., S. N. & Jalil, I. (2010), 'Software risk assessment visualization tool design using probabilistic inference models', *Information Technology (ITSim)* .

Pei-Chi, C. ; Ching-Chin, C. & C., C.-Y. (2012), 'Software project team characteristics and team performance: Team motivation as a moderator', *Software Engineering Conference (APSEC)* .

Pressman, R. S. (2006), *Engenharia de Software*, 6 edn, McGrawHill.

qiu Liu, Y., Zhou, C. & Zhang, Y. (2012), 'Coordinate preference dea method of software project risk assessment', *Inernational Conference on Communication Systems and Network Technologies (CSNT)* pp. 675 – 678.

Rausand, M. & Høyland, A. (2004), *System reliability theory: models, statistical methods, and applications*, Wiley-IEEE.

Report, T. (2010), The standish group international, inc. chaos summary, Technical report.

Sadiq, M. ; Rahmani, M. . A. M. . S. J. (2010), 'Software risk assessment and evaluation process(sraep) using model based approach', *Proceedings of the International Conference on Networking and Information Technology (ICNIT)* pp. 171 – 177.

Sahner, R., T. K. & Puliafito, A. (1996a), *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*, Kluwer Academic Pub.

Schwaber, K. & Beedle, M. (2002), 'Agile software development with scrum', *NJ: Prentence Hall* .

Shahzad, B. & S., A. (2010), 'Risk identification, mitigation and avoidance model for handling software risk', *CICSyN* pp. 1 – 196.

Silva, B. & et al. (2013), 'Astro: An integrated environment for dependability and sustainability evaluation', *Sustainable Computing: Informatics and Systems* pp. 1 – 17.

Solutions, S. (2006), 'mprime: solução integrada de gestão de riscos'. www.cin.ufpe.br/ suppera/mprime.php.

Somerville, I. (2011), *Engenharia de Software*, 9 edn, Addison-Wesley.

Sousa, E. & et al. (2012), 'Maintenance policy and its impact on the performability evaluation of eft systems', *International Journal of Computer Science, Engineering and Applications (IJCSEA)* .

Stewart, W. (1994), *Introduction to the numerical solution of Markov chains*.

Tang, A.-G. & long Wang, R. (2010), 'Software project risk assessment model based on fuzzy theory', *Computer and Communication Technologies in Agriculture Engineering (CCTAE)* .

Tracy, H., S. B. J. V. & David, W. (2008), 'The impact of staff turnover on software projects: The importance of understanding what makes software practitioners tick', *Proceedings of ACM SIGMIS CPR* .

Trivedi, K. S. (2001), *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, 2 edn, John Wiley and Sons.

Trivedi, K. & et al. (2009), 'Dependability and security models', *Design of Reliable Communication Networks (DRCN)* .

Vesely, W. & Roberts, N. (1987), *Fault tree handbook*, Nuclear Regulatory Commission.

Watson, J. F. & A., A. (1991), 'Applying generalized stochastic petri nets to manufacturing systems containing nonexponential transition functions', *In SMC* **21**.

Wattanapokasin, W. & Rivepiboon, W. (2009), 'Cross-cultural risk assessment model', *ICSPS* .

Yan, H. & Yu-feng, Z. (2011), 'Statistical prediction modeling for software development process performance', *Communication Software and Networks (ICCSN)* .

Zowghi, D. & Nurmuliani, N. (2002), ‘A study of the impact of requirements volatility on software project performance’, *Software Engineering Conference* .