STER: Uma Estratégia de Testes para Sistemas Reativos

Daniele Constant Guimarães

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

STER: Uma Estratégia de Testes para Sistemas Reativos

Daniele Constant Guimarães[1]
19 de Outubro de 2005

Banca Examinadora:

- Profa. Dra. Eliane Martins (Orientadora)
  Instituto de Computação – UNICAMP
- Prof. Dr. Edgar Toshiro Yano
  Instituto Tecnológico de Aeronáutica – ITA
- Prof. Dr. Ricardo de Oliveira Anido
  Instituto de Computação – UNICAMP
- Profa. Dra. Maria Cecília C. Baranauskas (Suplente)
  Instituto de Computação – UNICAMP

[1] Trabalho financiado parcialmente pela Capes.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Miriam Cristina Alves – CRB8a / 859

Guimarães, Daniele Constant.
G947s STER: uma estratégia de testes para sistemas reativos / Daniele Constant Guimarães -- Campinas, [S.P. :s.n.], 2005.

Orientadora: Eliane Martins.

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Software - Testes. 2. Avaliação de riscos. 3. Engenharia de software. I. Martins, Eliane. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 19 de outubro de 2005, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Edgar Toshiro Yano
INSTITUTO TECNOLÓGICO DE AERONÁUTICA – ITA

Prof. Dr. Ricardo de Oliveira Anido
IC - UNICAMP

Profa. Dra. Eliane Martins
IC - UNICAMP

STER: Uma Estratégia de Testes para Sistemas Reativos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Daniele Constant Guimarães e aprovada pela Banca Examinadora.

Campinas, 19 de outubro de 2005.

Profa. Dra. Eliane Martins
Instituto de Computação, Unicamp
(Orientadora)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

À minha família

"*Para que um grande sonho*
se torne realidade,
primeiro você precisa
de um grande sonho".
(Hans Seyle)

Agradecimentos

Gostaria de agradecer em primeiro lugar a Deus, por tudo que Ele me permitiu alcançar, pela força e perseverança concedida para a realização deste trabalho.

Gostaria de agradecer aos meus pais, Ângela e Eduardo, e às minhas irmãs, Nívea e Lícia, pelo amor, carinho, paciência, compreensão e apoio neste período.

Gostaria de agradecer à minha orientadora, Profa. Dra. Eliane Martins, sem a qual a realização deste trabalho não seria possível, pela confiança depositada em mim, pelo apoio e compreensão nos momentos difíceis.

Gostaria de agradecer à Ana Maria Ambrósio e à Maria de Fátima Mattiello Francisco pela ajuda e colaboração ao longo deste trabalho.

Gostaria de agradecer também a todos os meus amigos que acompanharam a minha caminhada, que contribuíram de forma direta ou indireta para a conclusão deste trabalho, que me ajudaram e apoiaram pelas cidades onde passei. A vocês meus amigos de casa, da pensão, de festas, de estudos, de passeios, de trabalho, de alegrias e tristezas, de “buteco”, aos meus amigos que apesar da pouca convivência e de quase nunca nos falarmos sempre me apoiaram. Aos amigos que vejo todo dia e àqueles que faz anos que não vejo...

Enfim, agradeço a todos vocês que me deram forças quando eu não a tinha mais, que me deram colo quando eu me sentia sozinha, que me fizeram prosseguir quando tentei desistir, que brigaram comigo quando foi necessário, que me fizeram sorrir quando eu queria chorar.

A todos vocês, o meu MUITO OBRIGADA!

Resumo

Alguns dos sistemas reativos existentes no mercado devem funcionar com um alto grau de confiabilidade, pois se eles falharem podem ocorrer perdas humanas ou financeiras significativas. A fim de minimizar as conseqüências das falhas, esses sistemas devem ser testados de forma que o maior número possível de falhas sejam detectadas. Para tanto, devem ser adotadas técnicas de teste automatizadas.

Na academia foram desenvolvidas diversas técnicas de geração automatizada de casos de testes usando métodos formais. Entretanto, essas técnicas ainda não são amplamente adotadas pela indústria, por vários motivos. Entre eles estão a dificuldade que os profissionais têm para usar métodos formais e a grande quantidade de casos de testes que podem ser gerados usando essas técnicas.

Este trabalho apresenta a STER, uma estratégia que tem como objetivo auxiliar nos testes de sistemas reativos e que pode ser usada sistematicamente pela indústria. Ela (i) faz uso de artefatos UML que podem ser produzidos na fase de análise tais como casos de uso e diagramas de seqüência, (ii) é baseada na especificação formal do sistema na forma de máquina finita de estados e (iii) usa análise de riscos para encontrar as partes mais críticas e vulneráveis do sistema.

Para encontrar quais são essas partes, e garantir que elas estão funcionando de acordo com o que foi especificado, foi definida uma técnica de análise de riscos para os testes que: (i) leva em consideração o que é importante para os usuários do sistema e (ii) permite concentrar os esforços de teste para reduzir o risco de ocorrerem falhas nessas partes.

Abstract

Some of the reactive systems must have a high degree of confiability. If they fail, significant humans or financial lost can occurs. To mitigate the consequences of these fails, this kind of systems needs to be tested to find the most fails as possible. One way to achieve this, is using automatic testing techniques.

Some automatic test case generation techiniques using formal methodos were delevolped by the academy. But they are not widely used by industry. There are several reasons for that. Between them are the dificulty to use formal methods and the huge number of test cases that can be generated using these techiniques.

This woks presents the STER, an strategy to help reactive systems testing. Also it can be used systematically by the industry. It (i) uses UML artifacts that can be produced in analysis phase like use cases and sequence diagrams, (ii) is based on system formal specification using finite state machine and (iii) uses risk analysis to find out the most critical and vulnerable parts of the system.

To find out these parts and to assure that they work as it was specified, a risk analysis technique for testing was defined. This technique considerates what is important to the system users and allows to concentrate an test effort to reduce the risk of a fail acurrence in these parts.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os sistemas reativos recebem estímulos vindos do ambiente onde se encontram e enviam respostas a ele. Alguns exemplos de sistemas reativos são: aplicações *web*, sistemas de tempo real, vídeo *games* e sistemas de controle de satélites e aviões, entre outros. Os sistemas não reativos como os sistemas batch, por exemplo, recebem apenas um estímulo inicial e a partir daí executam suas operações sem interagir com o mundo real.

Entre os sistemas reativos existem os que interagem com o mundo real controlando máquinas e/ou processos e que devem funcionar com um alto grau de confiabilidade. Normalmente, esses sistemas são complexos, concorrentes e distribuídos. Além disso, muitos deles lidam com um número de combinações de entrada que pode ser considerado infinito.

Podem ocorrer perdas significativas, humanas ou financeiras, se eles falharem. Portanto, idealmente, esses sistemas devem ser livres de falhas (fault). Por falha entende-se: "incorreção em um passo, processo ou definição de dados. É a manifestação no software de um erro cometido pelo desenvolvedor" [Martins, 2003].

Isso quer dizer que eles deveriam ser testados exaustivamente a fim de garantir que nunca falharão. Contudo, dependendo do seu tamanho e complexidade, o uso de testes exaustivos é impraticável [Pressman, 1997, cap 16]. Isso significa que ele pode conter falhas mesmo depois de ter sido testado. Por isso, esses sistemas devem ser testados de forma a encontrar o maior número possível de falhas.

Aumentar a quantidade de falhas encontradas durante os testes e, conseqüentemente, reduzir o número de falhas encontradas depois do sistema ter sido entregue torna-o mais confiável. Para tanto, devem ser adotadas técnicas para geração de casos de teste automatizadas. Ou seja, técnicas de teste que usam ferramentas para a geração e seleção de casos de testes [Martins+, 2004].

Na academia foram desenvolvidas diversas técnicas de geração automatizada de casos de testes. Muitas delas usam especificações formais tais como máquina finita de estados, redes de Petri, Statecharts, SDL, entre outras. Entretanto, essas técnicas ainda não são amplamente adotadas pela indústria. Entre os vários motivos existentes estão: a dificuldade que os profissionais têm para usar métodos formais e a grande quantidade de casos de testes que podem ser gerados usando essas técnicas [Jagadeesan+, 1997a], [Lai+, 1995].

Visando mitigar esses e outros problemas, foi desenvolvida a STER, uma eStratégia de TEstes para sistemas Reativos. Uma estratégia, segundo Pressman [Pressman, 1997, cap 17], "integra os métodos de projeto de casos de teste em uma série de passos bem planejados que resultam na construção de um software de sucesso".

A STER é uma estratégia para geração de casos de teste que pode ser usada sistematicamente pela indústria, pois é baseada na especificação do sistema, faz uso de artefatos UML produzidos na fase de análise e permite automatização. Essa estratégia é apoiada pela ConDado [Martins+, 1999], uma ferramenta para geração automatizada de casos de teste.

Nas seções que se seguem são mostrados a motivação deste trabalho, seus objetivos, contribuições e a organização da dissertação.

1.1 Motivação

A indústria possui seus próprios métodos de geração de casos de teste que são, normalmente, de natureza *ad-hoc*, baseados na experiência do testador e no uso de especificações informais. Sendo que, estas últimas podem ser ambíguas e causar erros de interpretação. Além disso, existem outros problemas que podem afetar a execução dos testes. Alguns deles, citados em Lai et al [Lai+, 1995], são:

- Dificuldade em se determinar o número de testes que devem ser executados para que o sistema possa ser considerado confiável. Principalmente quando se trata de sistemas complexos, porque neste caso, pode ser gerado um número grande de casos de teste.
- Dificuldade em se determinar quando os testes devem ser finalizados. Nem sempre é possível executar todos os testes disponíveis, pois, em geral, o tempo alocado para essa atividade é pequeno e ela é executada no fim do ciclo de desenvolvimento.
- Tendência dos testadores inexperientes em executar os mesmos caminhos no software não verificando nenhuma funcionalidade adicional do sistema.
- Alto custo dos testes que podem corresponder a mais de 50% do custo total de desenvolvimento do sistema.

O uso de métodos formais pode ajudar a contornar estes problemas contribuindo para a sistematização dos testes. No entanto, alguns aspectos devem ser observados [Jagadeesan+, 1997a], [Lai+, 1995]:

- **Métodos formais:** o uso de métodos formais auxilia na descoberta de falhas ainda na fase de projeto através da verificação da especificação do sistema. Esses métodos são baseados em teorias matemáticas, o que torna a especificação mais precisa e permite que ela seja verificada formalmente. Entretanto, eles são difíceis de serem usados na prática, pois nem todos os profissionais têm o conhecimento necessário para usá-los.

  Portanto, para que possam ser facilmente entendidos e usados, os testes baseados em métodos formais devem evitar o uso de teorias e modelos matemáticos complexos. Além disso, devem tratar problemas práticos como, por exemplo, encontrar o maior número de falhas com o mínimo de recursos e revelar as partes mais críticas do sistema.

- **Especificação estável:** Em muitos sistemas somente algumas funções ou aspectos são (ou podem ser) especificados no início. Além disso, os requisitos mudam ao longo do tempo: o sistema final desejado raramente é claro e completamente definido no início do desenvolvimento.

  Por isso, os testes baseados em métodos formais devem permitir que sejam usadas as partes da especificação do sistema que já estão finalizadas.

- **Partes críticas:** algumas partes do sistema são mais críticas que outras. Falhas nessas partes podem levar a perdas significativas, enquanto falhas em partes não-críticas do sistema são mais triviais.

  Sendo assim, os testes baseados em métodos formais devem permitir identificar os pontos críticos do sistema para que eles sejam tratados com mais atenção.

- **Grande número de casos de teste:** os métodos de geração de casos de teste existentes podem gerar um grande número de testes, principalmente para sistemas complexos. Alguns exemplos são o método adotado pela ConDado [Sabiao, 1998] e a técnica de teste N+ descrita por Binder [Binder, 2000, cap. 7].

  Dessa forma, os testes baseados em métodos formais devem fornecer uma maneira de selecionar os casos de teste sem, no entanto, deixar de lado as partes mais críticas do sistema.

Como mostrado acima, existem alguns fatores que podem afetar a geração e execução dos casos de testes. O uso de métodos formais pode contornar esses fatores. Contudo, existem alguns aspectos que devem ser observados para que eles possam ser usados. A existência destes aspectos motivou a criação de uma estratégia de teste, a STER, que permite a geração automatizada de casos de teste com base em métodos formais.

1.2 Objetivos

A partir da necessidade da indústria em contar com um método de geração de casos de teste efetivo, propomos uma estratégia de testes para sistemas reativos que:

1. Pode ser usada de forma sistemática pela indústria
2. Gera um número de casos de teste que pode ser totalmente executado.
3. Gera casos de teste para as partes mais críticas do sistema
4. Gera casos de teste de forma automatizada usando ferramentas.

Para atingir seus objetivos a STER tem as seguintes características:

- É baseada na especificação formal do sistema, uma vez que a maioria das falhas encontradas num sistema pode ser descoberta verificando a especificação [Lai+, 1995].
- É baseada em artefatos UML geralmente produzidos nas fases de análise e projeto, permitindo assim, o reuso de muitos modelos já existentes. Os artefatos UML foram escolhidos porque têm se tornado cada vez mais populares entre os desenvolvedores e têm sido adotados por muitas indústrias de software.
- É baseada em análise de riscos, que mostra quais são as partes mais críticas e vulneráveis do sistema. O que permite concentrar os esforços de teste para reduzir o risco de ocorrerem falhas nessas partes.
- A geração de casos de teste é feita a partir de parte da especificação do sistema. Isso permite que o número de casos de teste gerados seja reduzido e que possam ser gerados casos de teste mesmo que o sistema não esteja completamente especificado.
- A geração de casos de teste é automatizada. Para tanto, é usada a ferramenta ConDado, que está descrita no capítulo 4.

1.3 Contribuições

As principais contribuições deste trabalho são resumidas a seguir e detalhadas no capítulo 8. São elas:

- Foi definida uma estratégia de teste, a STER, que permite a geração automatizada de casos de testes usando artefatos produzidos na fase de análise e que pode ser usada sistematicamente pela indústria.
- Foi definido, passo a passo, como os casos de teste podem ser gerados a partir da especificação (em casos de uso) do sistema.
- Foi definida, passo a passo, uma técnica de análise de riscos para os testes que leva em consideração a finalidade que o sistema tem para o usuário.

Além disso, como conseqüência deste trabalho, podem ser respondidas perguntas importantes como "Em que ordem os testes devem ser executados?", “Como escolher os casos de teste que devem ser aplicados?” e “Como saber a hora de parar de testar?” [Binder, 2000, cap. 8].

1.4 Organização da dissertação

Essa dissertação está organizada da seguinte forma:

Capítulo 2: Apresenta os principais conceitos da área de testes e os vários tipos de testes que serviram como base para este trabalho.

Capítulo 3: Descreve os trabalhos relacionados que influenciaram a criação da STER e apresenta uma revisão bibliográfica.

Capítulo 4: Trata da geração automatizada de testes.

Capítulo 5: Apresenta a estratégia de geração de casos de teste e a técnica de análise de riscos propostas e seus procedimentos de execução.

Capítulo 6: Apresenta o uso da STER em um estudo de caso real.

Capítulo 7: Mostra os resultados obtidos usando a STER.

Capítulo 8: Descreve as conclusões deste trabalho.

Capítulo 2

Fundamentos de Testes

Cada dia mais as empresas têm procurado atender às normas de qualidade de software a fim de evitar que ocorram perdas financeiras ou de clientes para concorrentes. Para checar se uma aplicação atende às normas de qualidade de software, ou seja, para verificar se ela atende aos requisitos funcionais e não funcionais especificados e se atende aos padrões e convenções pré-estabelecidos de desenvolvimento, devem ser usadas técnicas de verificação e validação (V&V) [Sommerville, 2001, cap.19]. A verificação demonstra que as saídas do sistema estão de acordo com o que foi especificado. Contudo, isso não garante que elas estejam corretas, pois o sistema pode ter sido especificado de forma errada. Já a validação demonstra que as saídas estão de acordo com os requisitos do cliente.

Entre as técnicas de V&V existentes está a verificação dinâmica. Essa técnica envolve a execução do sistema com o objetivo encontrar falhas no produto. Falhas estas, que podem ter sido introduzidas durante o processo de desenvolvimento do software. O uso de testes é uma das formas de verificar dinamicamente um software [Sommerville, 2001, cap.19].

Durante os testes um sistema é executado sob condições específicas para detectar diferenças entre os resultados obtidos e os esperados. O teste de software tem por objetivos: (i) mostrar que os requisitos funcionais e não-funcionais de um sistema estão sendo cumpridos e (ii) descobrir o maior número possível de falhas em um sistema no menor tempo e com o menor esforço possível [Pressman, 1997, cap. 16], [Jacobson+, 1992].

O teste, para ser bem-sucedido, deve revelar a presença de falhas em um software. Entretanto, ele é incapaz de mostrar a ausência delas [Binder, 2000, cap. 3], [Pressman, 1997, cap. 16]. Em outras palavras, excetuando-se algumas situações muito específicas, se não foram encontradas falhas durante os testes não se pode dizer que a implementação está correta. Só se pode dizer que nenhuma falha foi encontrada. A presença de falhas no software fica caracterizada quando o comportamento apresentado pelo software é diferente do comportamento esperado.

Uma solução para mostrar a ausência de falhas em um software seria o uso de testes exaustivos, onde todas as seqüências de testes possíveis são executadas. Entretanto, executar testes exaustivos é impraticável por alguns motivos: (i) pode ser impossível enumerar todas as combinações possíveis de entrada/saída, (ii) pode ser impossível enumerar todas as seqüências de execução de um software e (iii) o custo da atividade de testes, que pode representar mais de 50% do custo de desenvolvimento.

Existe uma grande variedade de métodos para geração de casos de teste. Esses métodos fornecem critérios que permitem assegurar que os testes possuem uma alta probabilidade de encontrar falhas no software. Eles têm como objetivo testar a estrutura do sistema (técnicas caixa branca ou estruturais) ou testar a sua funcionalidade (técnicas caixa preta ou funcionais). As técnicas de teste caixa branca envolvem os testes baseados na implementação do sistema com o objetivo de exercitar suas instruções. Enquanto as técnicas de teste caixa preta envolvem os testes baseados na especificação do sistema e têm como objetivo verificar se uma determinada função está sendo executada de acordo com a especificação.

Uma das formas de gerar casos de testes é usando uma estratégia de testes [Binder, 2000, cap. 3]. Ela deve conter testes de alto nível para validar as funções do sistema em relação aos requisitos do usuário [Pressman, 1997, cap. 17]. Como um exemplo de estratégia de teste, podemos citar a estratégia proposta por Pressman [Pressman, 1997, cap. 17]. Ela é baseada no ciclo de vida espiral para o desenvolvimento de sistemas. Para cada fase do ciclo de vida Pressman sugere que seja executado um tipo de teste. A estratégia começa com a execução do teste de unidade. Em seguida, são executados os testes de integração, de validação e, por último, o teste de sistema. Essa estratégia é mostrada na Figura 2.1 e descrita a seguir.

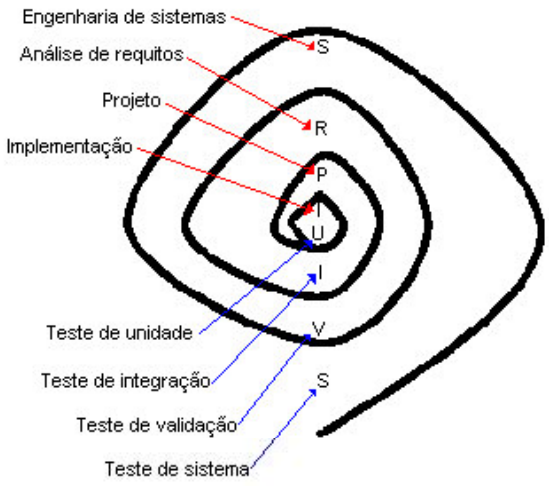

Figura 2.1: Estratégia de teste proposta por Pressman

- **Teste de unidade:** este teste é responsável por testar individualmente uma unidade (ou um componente) do sistema verificando se ela funciona corretamente. Depois de testadas individualmente as unidades são integradas para formar o sistema.
- **Teste de integração:** este teste, executado após o teste de unidade, tem o objetivo de verificar se os componentes do sistema funcionam corretamente depois de terem sido integrados.
- **Teste de validação:** executado após o teste de integração, este teste conta com a participação do usuário para garantir que os requisitos funcionais, comportamentais e de desempenho são satisfeitos.
- **Teste de sistema:** último teste executado no ciclo de desenvolvimento. Ele testa o sistema como um todo, isto é, depois dele ter sido combinado com outros elementos do sistema, como, por exemplo, hardware e bancos de dados.

Nas seções a seguir são definidos de forma geral os tipos testes que serviram como base para a STER, descrita no capítulo 5.

2.1 Teste de sistema

O teste de sistema é uma combinação de testes que tem como objetivo revelar falhas do sistema como um todo. Ele combina os testes de requisitos funcionais e de requisitos não funcionais para verificar se esses requisitos são satisfatórios quando o software é acoplado a outros elementos do sistema. Normalmente, os casos de teste executam ações típicas do usuário final do sistema.

O teste de requisitos funcionais verifica se as funcionalidades foram implementadas de acordo com a especificação usando técnicas de teste caixa preta. O teste de requisitos não funcionais verifica se a implementação satisfaz os requisitos não funcionais especificados, como por exemplo, os requisitos de desempenho e segurança. Alguns dos testes de requisitos não funcionais são: testes de stress, desempenho e tolerância à falhas.

2.2 Técnicas de teste caixa preta

As técnicas de teste caixa preta (ou funcionais) são baseadas na especificação do sistema e verificam se o sistema implementa os requisitos funcionais de acordo com a especificação. Essas técnicas são usadas para mostrar que as funções do software são operacionais; que a entrada é aceita de forma apropriada e a saída é produzida corretamente; e que a integridade das informações externas (arquivo de dados, por exemplo) é mantida [Pressman, 1997, cap. 16]. Resumindo, elas verificam o que o sistema faz, não como ele faz, ou seja, verificam a funcionalidade e não a implementação do sistema [Sommerville, 2001, cap. 20].

Entre as técnicas de teste caixa preta estão:

- **Teste baseado em modelos:** para a execução deste teste deve ser criado um grafo representando o sistema e depois um conjunto de casos de teste que o cubra de forma que cada nó e cada relacionamento sejam percorridos. Um dos testes baseados em modelos é o teste baseado em máquinas finitas de estados (MFE), descrito na seção 2.3.
- **Teste baseado em partições de equivalência:** para a execução deste teste o domínio de entrada e/ou saída do sistema deve ser dividido em um número finito de classes ou partições de equivalência, cada uma representando um conjunto de dados. Além disso, devem ser consideradas partições válidas e inválidas. Os dados que estão em uma mesma partição devem revelar as mesmas falhas. Para a geração dos casos de teste devem ser selecionados um ou mais dados de cada partição e cada partição deve ser considerada pelo menos uma vez.
- **Teste baseado em análises de valores limites:** Este teste complementa o teste baseado em partições de equivalência selecionando valores que estão nos limites das partições, como, por exemplo, arquivo vazio ou valor uma unidade menor que o valor mínimo válido. O uso deste tipo teste é recomendado, pois uma grande quantidade de erros tende a ocorrer nas margens do domínio de entrada, apesar de ainda não se saber bem o porquê disso [Pressman, 1997, cap.16].

2.3 Teste de transição de estados

O teste de transição de estados monitora as mudanças de estado do sistema em teste [Murray+, 1997]. Este teste consiste em verificar se o mapeamento de entradas/saídas para uma dada transição e se a seqüência de execução do sistema estão de acordo com a especificação [Binder, 2000, cap. 7]. Entre as falhas que podem ser encontradas executando este teste estão:

- **Falhas nas transições:** quando uma transição selecionada é incorreta ou inexistente o estado resultante é incorreto, mas não é corrompido.
- **Falhas nos eventos:** quando um evento selecionado é incorreto ou inexistente uma mensagem válida é ignorada.
- **Falhas nas ações:** quando uma ação ativada é incorreta ou inexistente uma ação incorreta é executada em resposta a um evento.
- **Falhas nos estados:** quando um estado extra, inexistente ou corrompido é ativado o comportamento do sistema torna-se imprevisível.

No teste de transição de estados são usadas as máquinas (ou modelo) finitas de estados clássicas (MFE) ou estendidas (MFEE). Essas máquinas descrevem o comportamento de um sistema em resposta a estímulos internos ou externos. Para que possam ser usadas nos testes essas máquinas devem modelar a especificação corretamente, ser consistentes e representar de forma precisa os requisitos do sistema que será testado [Binder, 2000, cap. 7].

Uma máquina finita de estados clássica é um sistema cuja saída é determinada pelo estado corrente do sistema e pelas entradas fornecidas. Ela é uma abstração composta por um número finito de eventos (entradas), estados (condição de um objeto em um determinado momento; estados observados pelos usuários), ações (saídas e resultados que se seguem a um evento) e transições (relacionamento entre dois estados indicando que, quando um evento ocorre, o estado passa do anterior para o subseqüente). A representação gráfica dessa máquina é chamada de diagrama de estados. Nela, os estados são representados por nós e as transições por setas rotuladas. Tais rótulos representam os eventos e ações. Um exemplo é mostrado na Figura 2.2.

A MFEE é uma extensão da máquina finita de estados clássica. Além dos componentes da MFE, a máquina finita de estados estendida possui variáveis, condições de guarda e parâmetros. Os parâmetros representam os dados que serão passados de um estado para outro. As condições de guarda são condições que tem que ser satisfeitas para que uma transição seja disparada. E as variáveis são usadas nas condições de guarda representando algum dado. Esses componentes são mostrados no exemplo a seguir.

Esse exemplo, mostrado por Binder [Binder, 2000, cap.7], modela uma pilha. O diagrama de estados é mostrado na Figura 2.2. Essa pilha tem três estados: “Vazia”, “Com dados” e “Cheia”. As funções ”Empilhar” e “Desempilhar” são modeladas como eventos e estão associadas às transições. As ações, que são associadas aos eventos e são representadas após a “/”, são: “ExceçãoPilhaVazia”, “ExceçãoPilhaCheia”, “retorna topo(x)”. Os “x” representam os parâmetros. As variáveis são: “n”, que representa o tamanho atual da pilha e “max", que representa o tamanho máximo da pilha. As condições de guarda ficam entre “[]” e são “[n = = 1]”, “[n>1]”, ”[n< Max-1]” e “[n = = Max-1]”.

A pilha começa no estado “Vazia”. Se for disparado o evento “Desempilha”, a ação “ExceçãoPilhaVazia” é executada. Se for disparado o evento “Empilha”, o dado “x” é empilhado e a máquina passa para o estado “Com dados”. Quando a máquina está no estado “Com dados” e chega um evento “Empilha” são verificadas as condições de guarda. Se o tamanho atual da pilha for menor que o tamanho máximo - 1 (representado pela condição de guarda [n < max - 1]), então o dado é empilhado e a máquina continua no estado “Com dados”. Se o tamanho atual da pilha for igual ao tamanho máximo - 1 (representado pela condição de guarda [n = = max - 1]), então o dado é empilhado e a máquina passa para o estado “Cheia”. A mesma lógica deve ser usada para os outros eventos.

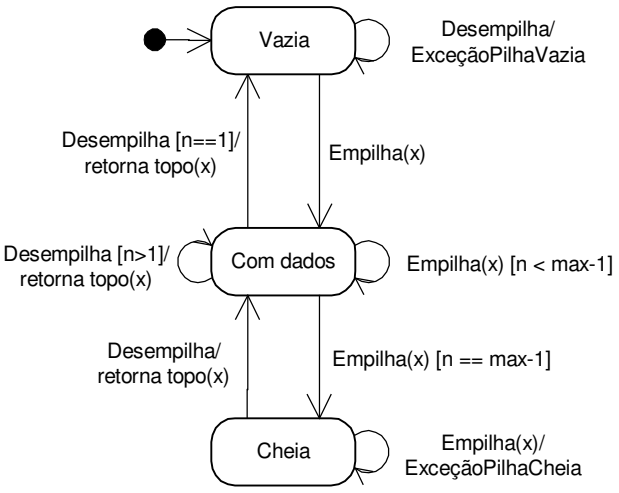

Figura 2.2: Diagrama de estados

2.4 Teste baseado em riscos

Os testes baseados em riscos levam em consideração os riscos associados ao software para evitar que ocorram falhas críticas quando o sistema estiver sendo executado. Tradicionalmente são executados testes baseados em riscos. Entretanto, eles são feitos de forma *ad-hoc* e com base na experiência do testador [Amland, 2000]. Estes testes devem ser feitos com a ajuda de uma análise de riscos, que consiste em identificar e avaliar os riscos do sistema. Dessa forma, as partes mais críticas do sistema podem ser testadas com maior cuidado mesmo que existam restrições de tempo ou dinheiro [Jagadeesan+, 1997b]. A análise de riscos será descrita com mais detalhes na seção 5.2.

Riscos são condições que, se ocorrerem, podem resultar em perdas ou causar algum impacto indesejado como, por exemplo, a produção de resultados incorretos ou a execução de transações não autorizadas. Quando se fala em riscos assume-se que existe a possibilidade de ocorrer uma perda. A situação de risco sempre existe, mas a perda pode não ocorrer. Os riscos NÃO podem ser eliminados, entretanto, sua ocorrência e/ou o impacto da perda podem ser reduzidos.

Em software, os riscos estão associados ao desenvolvimento e instalação do sistema e se ocorrerem podem afetar seu comportamento [Perry, 1995, cap. 2]. Existem dois tipos de riscos associados ao software: os genéricos, que são inerentes a todos os sistemas, e os específicos, que só podem ser identificados por pessoas que conhecem bem o sistema [Pressman, 1997, cap. 6]. Identificar os riscos é o primeiro passo para evitá-los quando possível e controlá-los quando necessário.

2.5 Teste baseado em casos de uso

Um caso de uso (CdU) representa as funcionalidades do sistema mostrando as interações entre ele e os atores externos. Um ator representa um usuário do sistema que pode ser uma pessoa ou um estímulo vindo do ambiente. Os casos de uso contêm as tarefas que são importantes do ponto de vista do ator. Além disso, são fáceis de entender e têm uma estrutura simples. Por isso, eles são uma boa fonte para os requisitos de testes de sistemas.

Apesar de fornecer dados que são necessários para os testes de sistema, como por exemplo, as interações existentes entre o sistema e os atores, os CdUs da UML (Unified Modeling Language) não mostram com que freqüência essas interações ocorrem nem o seu conteúdo. E essas informações também são importantes para os testes [Binder, 2000, cap. 8]. Para incluir estes itens, Binder [Binder, 2000, cap. 8] propõe que os CdUs sejam estendidos. As extensões propostas são:

1. **Definição do domínio de cada parâmetro (variável) que participa do caso de uso:** O domínio das variáveis é definido atribuindo um conjunto de valores válidos e inválidos para cada uma delas.
2. **Especificação dos relacionamentos requeridos de entradas/saídas entre as variáveis dos casos de uso:** Mostra a relação de dependência entre as entradas e as saídas produzidas para cada caso de uso.
3. **Restrições seqüenciais e dependências entre os casos de uso:** Levam em consideração as restrições e dependências em relação à execução de um caso de uso.
4. **Definição da freqüência relativa de uso para cada caso de uso:** A freqüência relativa de uso retrata o uso que o ator faz do CdU. O valor da freqüência atribuído para um CdU mais usado deve ser maior do que o valor atribuído para um menos usado.

2.6 Considerações finais

Neste capítulo foi mostrado que testar um sistema é uma forma de checar se ele atende às normas de qualidade de software, o que é uma preocupação para as empresas atualmente. Além disso, foi visto que, para cada fase do desenvolvimento de um software existe um tipo de teste adequado.

Este trabalho propõe uma estratégia de teste, baseada em técnicas de teste de caixa preta, que se aplique à fase de testes de sistema uma vez que o objetivo é verificar se as funções implementadas pela aplicação estão de acordo com os requisitos especificados.

Foram apresentados brevemente os tipos de teste aplicáveis ao escopo deste trabalho. São eles: testes de sistema e de caixa preta, que incluem os testes de transição de estados, testes baseados em riscos e testes baseados em casos de uso.

O teste de transição de estados foi escolhido porque ele permite monitorar as mudanças de estado do sistema em teste verificando se a seqüência de execução do sistema está de acordo com a especificação. Além disso, a máquina finita de estados estendida usada neste teste serve como base para a geração automatizada de testes pela ferramenta ConDado, descrita no capítulo 4.

O teste baseado em riscos, auxiliado por uma análise de riscos, foi usado porque permite a redução do número de casos de teste gerados sem, no entanto, deixar de gerar testes para as partes mais críticas do sistema, do ponto de vista do ator.

Os casos de uso fazem parte do escopo do trabalho porque eles têm sido amplamente usados para especificar os sistemas. Além disso, eles fornecem requisitos para os testes de sistemas permitindo assim, que os testes possam ser preparados mais cedo no processo de desenvolvimento. Isso reduz custos na fase de testes.

Capítulo 3

Estratégias de Testes de Sistemas

A STER é uma estratégia de geração de casos de teste para sistemas reativos baseada em técnicas de teste caixa preta que se aplica à fase de testes de sistemas. Ela usa testes baseados em riscos, em casos de uso e em modelos de estados. Isso permite que os casos de teste sejam gerados de forma automatizada e que a indústria use-a de forma sistemática, além de auxiliar nos testes das partes mais críticas do sistema. Neste capítulo trataremos dos trabalhos que serviram de inspiração para sua criação.

Após uma pesquisa bibliográfica sobre os tipos de testes e as estratégias de geração de casos de teste existentes foram encontrados vários trabalhos que tratam dos assuntos abordados pela STER. Alguns deles foram usados apenas como referência bibliográfica e são mostrados na seção 3.8. Outros influenciaram diretamente a criação da STER e serão descritos nas próximas seções. Na seção 3.9 são destacados os pontos desses trabalhos que influenciaram a criação da STER. São eles:

- O método SCENT [Ryser+, 2000] e a metodologia TOTEM [Briand+, 2002], onde a derivação de requisitos de testes é baseada em artefatos da UML.
- O processo de seleção de casos de teste baseado nas prioridades do usuário proposto por McGregor et al [McGregor+, 2000].
- A estratégia ETACS [Volpi, 2001], que foi selecionada por ser baseada em riscos.
- A técnica de teste baseada em riscos proposta por Amland [Amland, 2000].
- O processo padronizado TEST-Rx para testes de aplicações cliente-servidor proposto por Mosley [Mosley, 2000, cap. 4].
- As táticas de teste (ou plano de teste) definidas por Perry [Perry, 1995, cap.3].

3.1 SCENT (SCENario-Based Validation and Test of Software)

O método SCENT [Ryser+, 2000] é baseado em cenários[2] e foi desenvolvido para:

- Validar e verificar requisitos através da formalização dos casos de uso descritos em linguagem natural.
- Desenvolver casos de teste percorrendo caminhos no *statecharts*.

Esse método representa os casos de uso de duas formas: (i) em linguagem natural narrativa e (ii) em uma notação gráfica usando *statecharts*.

Primeiro são usados os CdUs em linguagem natural, pois eles são mais fáceis de criar e de entender. Entretanto, como eles podem ser ambíguos, vagos e imprecisos dando margem a várias interpretações diferentes, eles são transformados em *statecharts*. Essa formalização torna possível descobrir omissões e contradições na especificação. Apesar de ser mais difícil de entender, o *statecharts* possibilita a geração automatizada de casos de teste.

Esse método:

- Usa casos de uso para: (i) levantar e documentar os requisitos, (ii) descrever a funcionalidade e especificar o comportamento do sistema e (iii) para validar o sistema em desenvolvimento enquanto ele está sendo desenvolvido.
- Apresenta um procedimento para levantar, criar e estruturar os casos de uso durante a fase de análise.
- Explicita e mostra as dependências entre os casos de uso usando um diagrama de dependência que foi definido pelos próprios autores.
- Anota o *statecharts*, quando necessário, para fornecer as informações necessárias para geração dos casos de teste. Essas anotações podem ser, por exemplo, pré e pós-condições e valores de dados e requisitos de desempenho.
- Deriva os casos de teste de percorrendo os caminhos do *statecharts* e do diagrama de dependência.

[2] As definições de cenários e casos de uso que são adotadas pela SCENT diferem das definições adotadas pela UML. Usaremos neste trabalho as definições mostradas nas seções 2.5 e 5.1, que também se aplicam à SCENT. A partir desse ponto os cenários da SCENT serão tratados como casos de uso.

3.2 ETACS (Estratégia de Teste de Software para Ambiente Cliente-Servidor)

A ETACS [Volpi, 2001] foi desenvolvida para testar aplicações cliente-servidor. Essa estratégia usa casos de uso para capturar e definir os requisitos funcionais do sistema. Ela possui 6 etapas, sendo que, cada uma dessas etapas possui uma série de atividades.

1. **Etapa Inicial:** nesta etapa são geradas as informações iniciais sobre o projeto de sistema que deverá ser desenvolvido. Entre elas estão as estimativas de custo para os testes.
2. **Plano de teste:** entre as atividades desta etapa estão: a identificação dos requisitos do sistema, avaliação dos riscos e a definição das prioridades. As atividades de avaliação dos riscos e de definição de prioridades são responsáveis por identificar os casos de uso que representam maior risco para o sistema.

   Para calcular a prioridade de cada um dos casos de uso, primeiro são avaliados os riscos. Isso é feito analisando alguns itens ponderados, tais como, importância de cada CdU para o sistema e freqüência de uso do CdU. Depois é atribuído um grau de importância para cada um deles. Esse grau de importância varia de 1 a 5, sendo 5 o mais crítico.

   Em seguida é feito o cálculo da prioridade que consiste em multiplicar o grau de importância do item pelo seu peso e somar os valores encontrados por caso de uso.

   A classificação final da prioridade dos casos de uso depende do valor obtido na soma. Quanto maior o valor obtido, maior risco ele representa para o sistema.
3. **Projeto de casos de teste:** nesta etapa os casos de teste são extraídos a partir dos casos de uso e são selecionados os que deverão ser aplicados.
4. **Execução de casos de teste:** os casos de teste são executados levando-se em consideração o ciclo de vida iterativo de desenvolvimento. Em cada fase é executado o teste apropriado àquela fase.
5. **Avaliação dos casos de testes implementados:** os casos de teste são avaliados usando métricas pré-estabelecidas.
6. **Documentação dos casos de testes e resultados:** os casos de teste devem ser documentados para uso futuro.

3.3 TOTEM (Testing of Object-Oriented Software SysTEms with the UML)

O objetivo da TOTEM [Briand+, 2002] é apoiar a derivação dos requisitos de testes funcionais. Os requisitos de teste são uma especificação precisa do que deve ser testado. Para derivar esses requisitos são usados os seguintes artefatos UML produzidos na fase de análise: diagramas e descrições de casos de uso, diagramas de interação e diagramas de classe. Esses últimos são compostos das classes do domínio da aplicação e dos dicionários de dados que descrevem cada classe, método ou atributo. Os passos da TOTEM são mostrados na Figura 3.1.

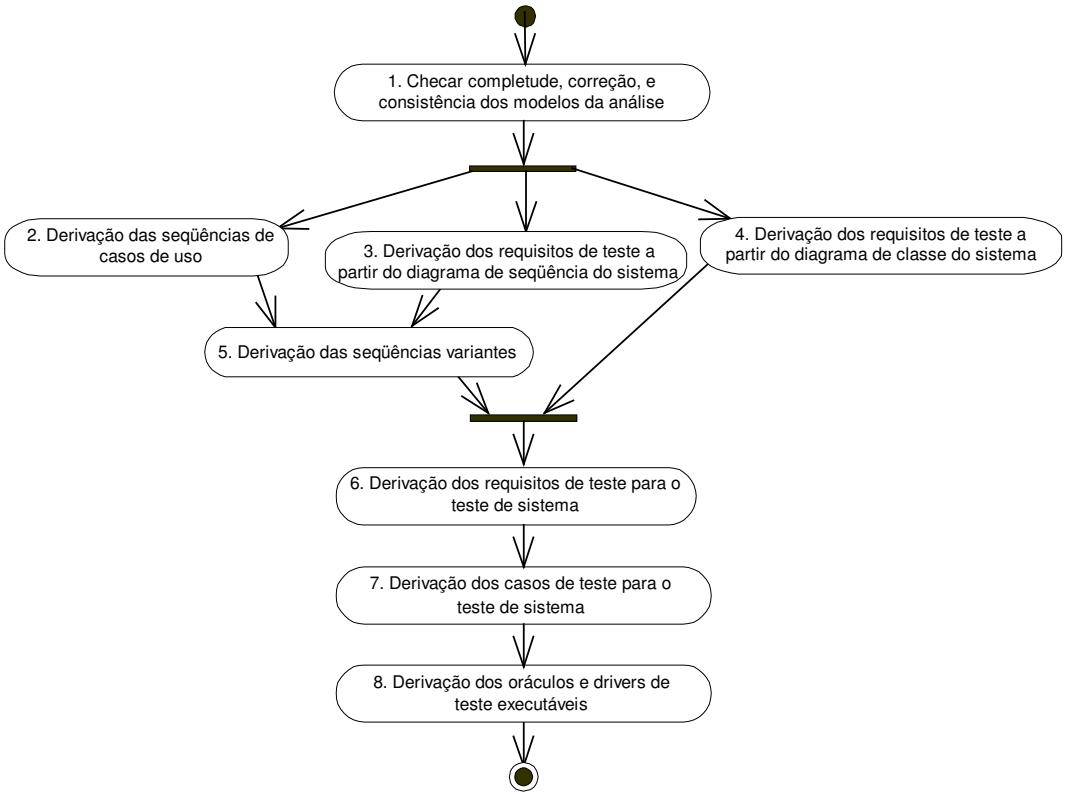

Figura 3.1: Passos da TOTEM

Depois de garantir que os modelos da análise são testáveis (passo 1), os requisitos de teste são derivados a partir de diferentes artefatos (passos 2 a 5). Em seguida, esses requisitos são fundidos em um único conjunto de requisitos de teste (passo 6), retirando-se as redundâncias e combinando os requisitos de teste em um plano de teste. E por fim são derivados os casos de teste e os códigos para os oráculos [Binder, 2000, cap.18] embutindo-os em *drivers* de teste executáveis (passos 7 e 8).

Para a geração dos requisitos de teste são usados os passos 2, 3 e 5 que serão detalhados a seguir.

- **Passo 2 – Derivação das seqüências de casos de uso:** Normalmente, os casos de uso têm restrições e dependências em relação à sua execução. Isso significa que eles devem ser executados em uma determinada ordem. O uso do diagrama de atividades auxilia na visualização dessa dependência e na determinação das seqüências de CdUs. Uma forma de obter essas seqüências é percorrer os caminhos do diagrama de atividades. Para a geração das seqüências de CdUs, levando em consideração essa dependência, a TOTEM usa os casos de uso estendidos descritos em [Binder, 2000, cap. 8] e mostrados na seção 2.5. A dependência entre os parâmetros dos casos de uso também deve ser levada em consideração na hora de determinar as seqüências de CdUs.
- **Passo 3 – Derivação dos requisitos de teste a partir do diagrama de seqüência do sistema:** A derivação dos requisitos de teste é feita a partir do diagrama de seqüência do sistema. Primeiro esse diagrama é re-escrito como uma expressão regular e as condições para que um determinado caminho no diagrama de seqüência seja habilitado são escritas em OCL (Object Constraint Language) [Warmer+, 1999]. Essas condições são obtidas a partir das condições de guarda associadas aos caminhos do diagrama de seqüência. Em seguida, devem ser identificadas as seqüências das operações que serão executadas.
- **Passo 5 – Derivação das seqüências variantes:** As seqüências variantes são geradas a partir das seqüências de CdUs. Para cada seqüência de CdU existe um conjunto de condições que devem ser satisfeitas para que essa seqüência possa ser testada. Cada um desses conjuntos forma uma seqüência variante.

3.4 Processo de seleção de casos de teste baseado nas prioridades do usuário

Nem os testes caixa preta, nem os testes caixa branca levam em consideração a prioridade que os usuários dão para as funções utilizadas. Ou seja, as funções que são mais importantes para o usuário[3] ou aquelas que são usadas com mais freqüência são indiferentes para estratégias de teste como a SCENT [Ryser+, 2000] e a TOTEM [Briand+, 2002], entre outras. Considerar o uso que cada ator faz do sistema pode auxiliar na tarefa de descobrir, antecipadamente, as falhas que podem ser encontradas durante sua execução.

McGregor et al [McGregor+, 2000] apresenta um processo de seleção de casos de teste baseado nas prioridades do ator, que tem como objetivo testar os CdUs usados com mais freqüência. Para tanto, ele usa a freqüência relativa de uso (freqüência com que um CdU é acionado em comparação com os outros casos de uso) e a criticidade dos CdUs

[3] A partir desse ponto trataremos o usuário do sistema como um ator, uma vez que um ator representa um usuário do sistema.

(define o quanto o CdU é necessário para o sucesso da operação). Nesse processo, a criticidade é definida com base na experiência do testador e a freqüência relativa de uso é determinada de acordo com o uso que o ator faz do sistema.

Cada ator do sistema é descrito por um perfil, que permite obter informações relativas à freqüência de uso dos CdUs. Essa descrição é exemplificada na Tabela 3.1 usando um sistema bancário simples. As atividades do processo são:

1. **Definir o conjunto de atores do sistema.**
2. **Definir o conjunto de casos de uso incluindo a associação com os atores.**
3. **Para cada um dos atores do sistema:**
   a. Definir o peso do ator, caso seja necessário.
   b. Definir o nível de habilidade do ator.
   c. Definir os *Casos de Uso* e a *Freqüência Relativa de Uso* para cada um deles.
4. **Calcular a freqüência relativa de uso total:** *para cada um dos CdUs*, combinar os valores da freqüência relativa de uso encontrada para cada um dos atores do sistema em uma única freqüência relativa de uso usando uma média ponderada. A média ponderada é usada porque os atores podem ter pesos diferentes.
5. **Calcular a prioridade do caso de uso:** combinar a freqüência e a criticidade para cada CdU tirando uma média entre esses valores ou usando uma estratégia conservativa (onde o valor mais alto é escolhido por *default*).
6. **Alocar os casos de teste com base nas prioridades dos CdUs:** distribuir os casos de teste disponíveis de acordo com a prioridade do caso de uso. Para os CdUs de prioridade alta devem ser alocados mais testes do que para os CdUs de prioridade baixa.

Tabela 3.1: Descrição detalhada do ator *Portador de conta*

Ator: Portador de conta **Descrição:** Possui uma conta no banco e pode executar atividades relacionadas a ela. **Nível de habilidade:** Inexperiente **Perfil de uso:**	
Caso de uso	**Freqüência relativa de uso**
Fazer saque	Alta
Fazer depósito	Média

Essa técnica não altera o processo de seleção dos tipos de teste que serão usados nem mostra quantos casos de teste devem ser construídos. Ela mostra apenas quais são os casos de uso exercitados com mais freqüência pelos atores permitindo que os casos de teste possam ser alocados de acordo com a prioridade de cada um dos CdUs.

3.5 Técnica de teste baseada em riscos

Amland [Amland, 2000][4] descreve uma técnica de teste cujo objetivo é mostrar como os testes baseados em riscos podem apoiar o gerenciamento do processo de teste e auxiliar na tomada de decisões. Essa técnica é composta por 6 passos:

1. **Planejamento:** Criação do plano de teste, que deve conter uma estratégia de riscos. Essa estratégia é definida de acordo com o projeto e depende do tipo do sistema, do ambiente onde será ele usado e dos requisitos de qualidade e comerciais.
2. **Identificar indicadores de riscos:** Os indicadores de risco (j) podem ser, por exemplo, o tamanho de cada função, número de alterações desde a última entrega e freqüência relativa de uso, entre outros.

   Deve ser atribuído um valor numérico para cada um dos indicadores (i_j). Os valores usados são: risco baixo (1), risco médio (2), risco alto (3). Além disso, cada um dos indicadores possui um peso (w_j) associado a ele. O peso varia de 1 a 5, sendo o 5 o mais crítico.

   A probabilidade ($P(f)$) de ocorrer uma falha na função *f* é dada pela fórmula a seguir:

   $P(f) = \sum_{j=1}^{x} i_j * w_j$, onde i = [1..3], w = [1..5] e x é o número total de indicadores

3. **Identificar o custo de uma falha:** O custo que o cliente ou o fornecedor terá se uma falha ocorrer é identificado por eles em conjunto com o testador. O custo total ($C(f)$) se ocorrer uma falha na função *f* é dado pela média entre o custo para o cliente ($cc(f)$) se ocorrer uma falha na função *f* e o custo para o fornecedor ($sc(f)$) se ocorrer uma falha na função *f*.

   $$C(f) = \frac{sc(f) + cc(f)}{2}$$

4. **Identificar os elementos críticos:** depois de identificadas a probabilidade (passo 2) e o custo (passo 3), a exposição ao risco ($\mathrm{Re}(f)$) pode ser calculada usando a fórmula:

   $$\mathrm{Re}(f) = P(f) * C(f)$$

   O número obtido pode ser usado para ordenar e priorizar as diferentes funções.

[4] Nesta seção estão sendo usados os mesmos termos utilizados no artigo citado.

5. **Execução dos testes:** Depois da lista de funções priorizadas ter sido definida, começa a execução dos testes, onde é dada uma maior ênfase às funções mais críticas.
6. **Estimativas:** São usadas as métricas coletadas durante a primeira parte dos testes para prever o total de recursos que serão necessários e identificar as áreas críticas com base em indicadores de probabilidade.

3.6 TEST-Rx

Mosley [Mosley, 2000, cap. 4] descreve um processo padronizado de teste, o TEST-Rx, que serve como base para as atividades de teste de aplicações cliente-servidor. Entre outras coisas, ele garante (i) que as partes de maior risco do sistema são identificadas e podem ser testadas, (ii) fornece uma base para automação dos testes e (iii) evita testar partes do sistema que não são importantes ou que resultam em testes redundantes.

Esse processo consiste em uma série de passos que por sua vez são divididos em uma série de atividades. Os passos são:

1. **Montar a equipe de testes:** monta a equipe de testes para verificação e validação da implementação.
2. **Fazer a análise de riscos:** a análise de riscos é feita para identificar os componentes de maior risco do sistema.
3. **Estabelecer os objetivos dos testes:** estabelece as metas dos testes.
4. **Construir o plano de teste:** elabora um documento contendo os papéis dos testadores, o ambiente e os recursos necessários para os testes. Esse documento deve descrever também as estratégias e os casos de teste.
5. **Projetar e construir os casos de teste**: aplica técnicas de projetos de casos de teste para projetar e construir os dados do teste.
6. **Executar os testes de unidade e integração**
7. **Executar os testes de sistema e aceitação**
8. **Analisar e relatar os resultados dos testes**

Mosley [Mosley, 2000, cap. 4] propõe ainda, uma classificação para as falhas encontradas no sistema, que pode ser usada durante a análise de riscos. A classificação é mostrada na Tabela 3.2.

Tabela 3.2: Classificação das falhas segundo Mosley [Mosley, 2000, cap. 4]

Classificação	Descrição
Crítica	A falha resulta em um defeito do sistema ou de parte dele e não existem processos alternativos que permitam alcançar o resultado desejado.
Prioritária	A falha resulta em um defeito do sistema ou de parte dele. Não existe como fazer a unidade voltar a funcionar, entretanto, existem processos alternativos que permitem alcançar o resultado desejado.
Regular	A falha não resulta em um defeito do sistema, mas faz com que o sistema produza resultados incompletos, incorretos ou inconsistentes ou danifica a sua usabilidade.
Pouca importância	A falha não resulta em um defeito do sistema nem danifica a sua usabilidade. O resultado desejado é obtido facilmente contornando o defeito.
Exceção de qualidade	A falha é relacionada à estética do sistema ou à requisição para uma melhoria. Sua correção pode ser adiada ou ignorada.

3.7 Táticas de teste

Perry [Perry, 1995, cap. 3] define táticas de teste (ou plano de teste) como sendo um meio pelo qual a estratégia de teste é realizada. Ele propõe um processo de 8 passos para desenvolver essas táticas. Os passos são descritos a seguir.

1. **Adquirir e estudar uma estratégia de teste.**
2. **Determinar o tipo de projeto de desenvolvimento:** o tipo de projeto se refere à metodologia/ambiente onde o software será desenvolvido.
3. **Determinar o tipo de sistema:** o tipo de sistema se refere ao processamento que será executado pelo sistema. Por exemplo: se ele é um software de aquisição de dados ou de gerenciamento de base de dados, entre outros.
4. **Determinar o escopo do projeto:** refere-se à totalidade de atividades que serão incorporadas ao sistema que está sendo testado; o conjunto de especificações/requisitos do sistema que devem ser entendidos.
5. **Identificar os riscos:** divide os riscos em 3 categorias: riscos estruturais (riscos associados com a aplicação e os métodos usados para construí-la), riscos técnicos (riscos associados com a tecnologia usada na construção e operação do sistema) e riscos de tamanho (riscos relacionados à grandeza do software em todos os aspectos).
6. **Determinar quando os testes devem ser feitos:** devem ser feitos durante todas as fases do projeto[5].

[5] Perry [Perry, 1995] diz que os testes devem ser feitos ao longo do desenvolvimento, pois para ele os testes envolvem as técnicas estáticas (como por exemplo, revisões e inspeções) além das técnicas dinâmicas, como usado nesse trabalho.

7. **Construir o plano de teste do sistema:** deve ser elaborado um documento, chamado plano de teste, descrevendo quando e como o teste será feito. Esse plano deve conter, entre outras coisas, informações sobre o sistema que será testado, os objetivos e os riscos dos testes e quais são os testes que deverão ser executados.
8. **Construir o plano de teste de unidade:** cada uma das unidades/componentes do sistema deve ter seu próprio plano de teste.

Perry [Perry, 1995, cap. 3] mostra um método para calcular os riscos usando questionários, onde são identificados os riscos e as taxas e pesos associados a eles. Os riscos e as taxas são fornecidos como parte do processo de cálculo do risco. O peso é a indicação da importância relativa de cada risco em relação aos outros riscos.

Para preencher os questionários deve-se determinar os riscos, a taxa apropriada para o sistema que está sendo testado e por fim, calcular e acumular o placar do risco.

3.8 Outros trabalhos

Além das bibliografias descritas anteriormente, foram estudados outros trabalhos:

- Bertolino et al [Bertolino+, 2004a]: apresenta uma técnica para testes baseados em modelos que pode ser usada pela indústria. A especificação deve ser feita usando diagramas de seqüência e diagramas de estados da UML.
- Bertolino et al [Bertolino+, 2004b]: apresenta uma metodologia de testes para linhas de produtos usando uma extensão para os casos de uso.
- Farias [Farias, 2003]: apresenta um método de teste funcional para verificação de componentes baseados em artefatos UML.
- Chen [Chen, 2002]: apresenta uma técnica para seleção de casos de teste de regressão. Essa técnica é baseada na especificação do sistema e em análise de riscos para a seleção dos casos de teste.
- Wittevrongel et al [Wittevrongel+, 2001a]: apresenta a técnica SCENTOR, que fornece apoio específico ao e-business, para a geração de testes baseados em casos de uso usando diagramas de seqüência da UML.
- Regnell et al [Regnell+, 2000]: apresentada uma forma de integrar a modelagem usando casos de uso e os testes baseados em usos estatísticos.
- Bousquet et al [Bousquet+, 1999]: apresenta a ferramenta Lutess desenvolvida para teste de softwares reativos síncronos.

- Jagadeesan et al [Jagadeesan+, 1997a]: apresenta uma técnica para teste de conformidade de sistemas reativos síncronos.
- Rosenberg et al [Rosenberg+, 1996]: a Nasa/SATC (Software Assurance Technology Center) usa métricas para calcular os riscos em cada fase do ciclo de desenvolvimento e para projetar riscos futuros.
- Grabowski [Grabowski, 1994]: apresenta o método SAMSTAG que foi desenvolvido para a geração automática de casos de teste de conformidade baseado na especificação formal do sistema. As especificações de protocolos devem ser escritas em SDL (Specification and Description Language) e MSC (Message Sequence Charts).
- Ural [Ural, 1992]: apresenta vários métodos para geração de seqüências de teste a partir de especificações baseadas em máquinas finitas de estados.

3.9 Considerações finais

A STER foi criada tendo como base os trabalhos descritos acima. Da SCENT [Ryser+, 2000] foi usada a idéia de formalizar os casos de uso usando máquinas de estado. Ela foi usada como base também para gerar os CdUs e cenários do exemplo apresentado no Apêndice A, uma vez que eles não estavam disponíveis. Não foram usados: o diagrama de dependência apresentado, pois ele não faz parte da UML e o *statecharts* porque ele não é aceito pela ferramenta adotada, a ConDado.

Parte do processo de seleção de casos de teste baseado nas prioridades do usuário proposto por McGregor et al [McGregor+, 2000] foi usada para obter informações sobre a freqüência de uso dos CdUs do sistema, para que os casos de testes gerados levassem em consideração o uso que os atores fazem do sistema.

Para a definição prioridades dos casos de uso e a identificação daqueles que representam um maior risco para o sistema, a STER se baseou na ETACS [Volpi, 2001], na técnica de testes baseada em riscos proposta por Amland [Amland, 2000], na TEST-Rx [Mosley, 2000] e nas táticas de teste proposta por Perry [Perry, 1995, cap. 3]

A STER se baseou nos passos 2, 3 e5 da TOTEM [Briand+, 2002] para a geração dos casos de teste. Isso permite que sejam testados apenas os casos de uso que estão nas seqüências que contêm os CdUs de maior risco, o que diminui o número de casos de teste gerados. Além disso, o diagrama de seqüência usado para representar um cenário que pertence a um caso de uso serviu de base para gerar uma máquina de estados clássica para o cenário.

Capítulo 4

Geração automatizada de testes

A geração automatizada dos testes permite gerar um número maior de casos de teste tornando o sistema mais confiável. Pensando nisso, a STER usa, para a geração automatizada de casos de testes, a ConDado (CONtrole e DADOs) [Sabiao, 1998], uma das ferramentas desenvolvidas no contexto do projeto ATIFS (Ambiente de Testes baseado em Injeção de Falhas por Software) [Martins, 1995].

Esse ambiente dá suporte à geração e execução dos testes e à análise dos resultados obtidos. Ele foi desenvolvido em um projeto conjunto entre o Instituto de Computação da Universidade Estadual de Campinas (IC – UNICAMP) e a Divisão de Sistemas de Solo do Instituto Nacional de Pesquisas Espaciais (DSS – INPE).

Nesse capítulo serão apresentados: o ambiente ATIFS (no qual a ConDado está inserida), o MME (Modelador de Máquina de Estados – uma ferramenta de apoio) e a ConDado.

4.1 ATIFS (Ambiente de Testes baseado em Injeção de Falhas por Software)

O ATIFS é um ambiente que foi desenvolvido para dar apoio ao processo de teste. Ele: (i) auxilia na geração automatizada de casos de testes a partir de um modelo formal do sistema, (ii) permite a execução dos casos de teste gerados e (iii) facilita a análise dos resultados obtidos [Martins, 1995]. As ferramentas que o compõe são: MME, ConDado, GerScript, Fsofist e AnTrEx. Sua arquitetura é mostrada na Figura 4.1.

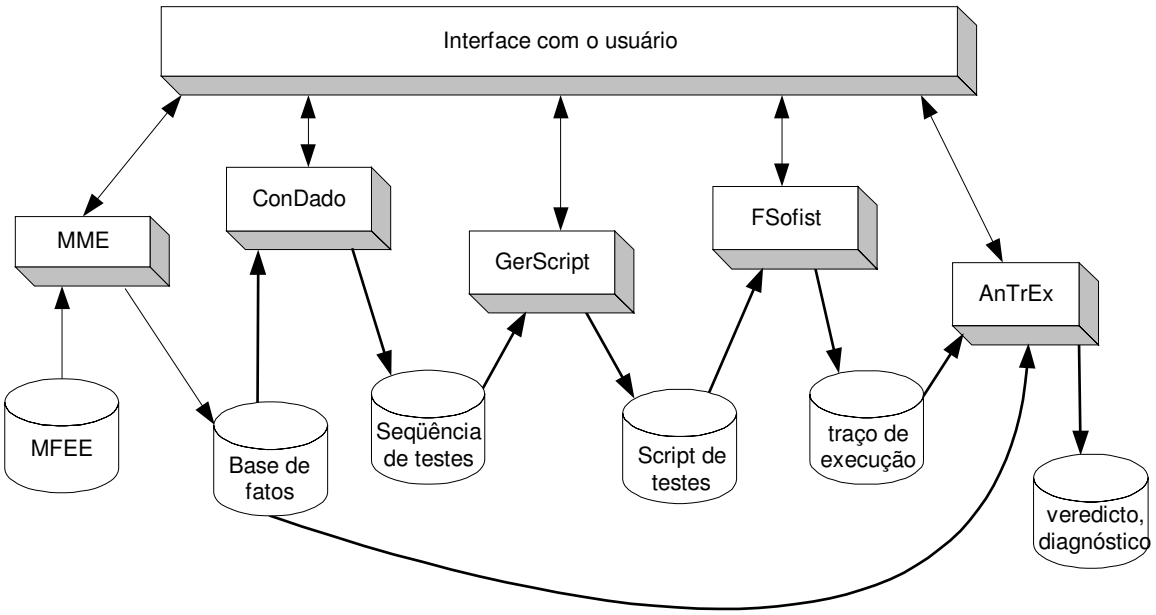

Figura 4.1: Arquitetura do ATIFS

O MME gera uma base de fatos em Prolog a partir de uma máquina finita de estados estendida (MFEE). A base de fatos gerada é a entrada para a ConDado que gera uma seqüência de testes. A partir dessa seqüência de testes a GerScript gera os scripts de testes que descrevem como a seqüência de teste deve ser realizada. Em seguida, a FSofist realiza a injeção de falhas por software usando para isso os scripts de teste que foram gerados pela GerScritpt. E por fim, a AnTrEx realiza a análise de traços, ou seja, a análise dos resultados obtidos.

Detalharemos a seguir as ferramentas MME e ConDado. As outras não serão detalhadas, pois não foram usadas neste trabalho.

4.2 MME (Modelador de Máquina de Estados)

A Figura 4.2 mostra a interface do MME [Melo, 2003], uma ferramenta desenvolvida pelo INPE/Natal dentro do escopo do projeto ATIFS. Essa ferramenta tem como objetivo automatizar a transformação da especificação descrita em máquina finita de estados estendida em uma base de fatos Prolog, que é a entrada para a ConDado.

Ela possui facilidades que permitem ao usuário modelar a MFEE de forma gráfica e sem muito esforço, através do uso de botões de estados e transições. A área 1 da Figura 4.2 contém as operações permitidas para os estados e transições. A máquina de estado é desenhada na área 2. E por fim, a área 3 permite a inclusão dos dados associados a uma entrada (transição), como por exemplo, seu nome, tipo e valor. A partir do diagrama criado é possível gerar a base de fatos em Prolog automaticamente.

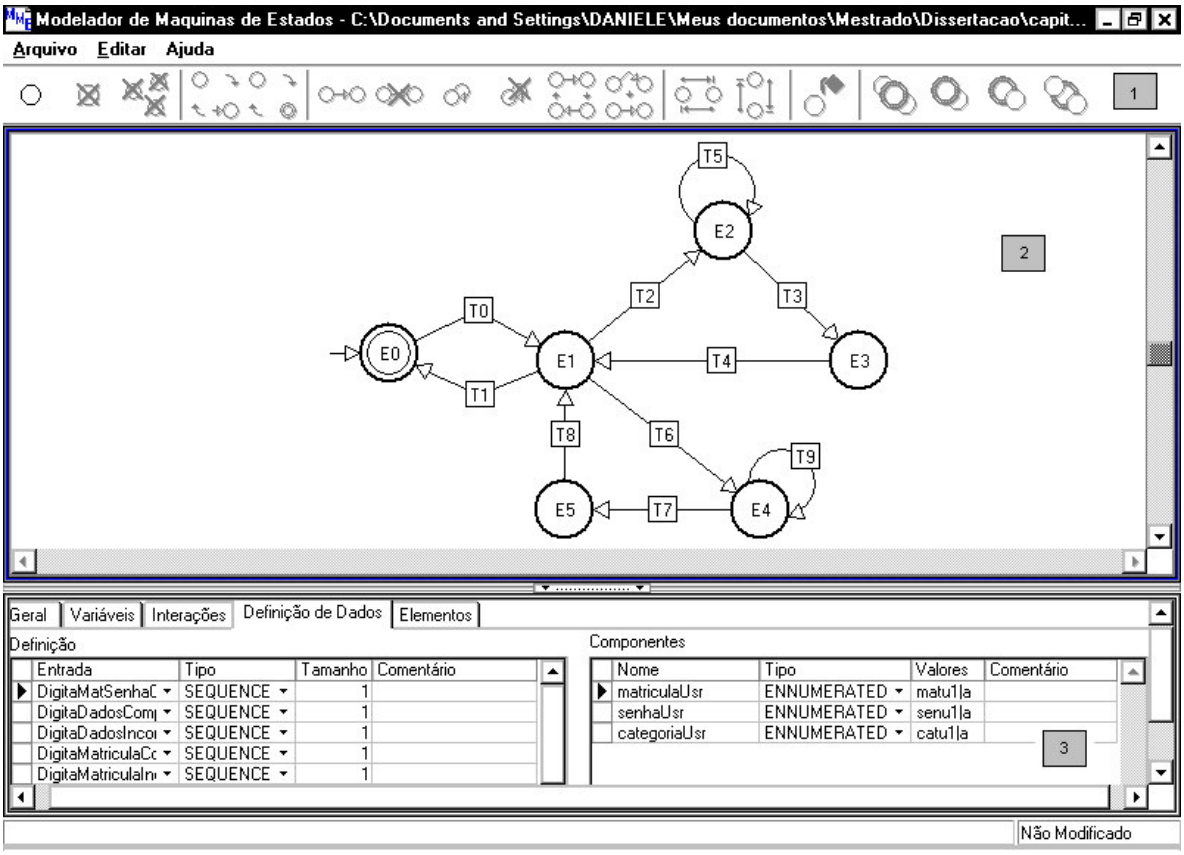

Figura 4.2: Modelador de Máquina de Estados

4.3 ConDado (Controle e Dados)

A ConDado é uma ferramenta que foi desenvolvida, inicialmente, para apoiar os testes de protocolos de comunicação baseado na especificação formal do mesmo com o objetivo de automatizar o processo de geração de casos de teste. Para tanto, são usadas algumas técnicas de teste caixa preta e o protocolo deve ser especificado usando máquina finita de estados estendida [Sabiao, 1998]. Os testes gerados combinam controle (representa os estados do protocolo e, as entradas e saídas decorrentes da mudança de estado) e dados dos parâmetros de interações de protocolos (descrevem o formato e o tipo dos dados trocados).

Para a geração dos casos de teste, os requisitos devem ser especificados usando uma máquina finita de estados estendida que depois é transformada em uma especificação de teste incluindo os dados e o controle. Os casos de teste são gerados a partir dessa especificação e das restrições levantadas pelo usuário, caso existam, como será descrito a seguir. Esses passos são mostrados na Figura 4.3. Os passos 1 e 2 são implementados pelo MME mostrado na seção anterior. O passo 3 é implementado pela ConDado e será detalhado a seguir.

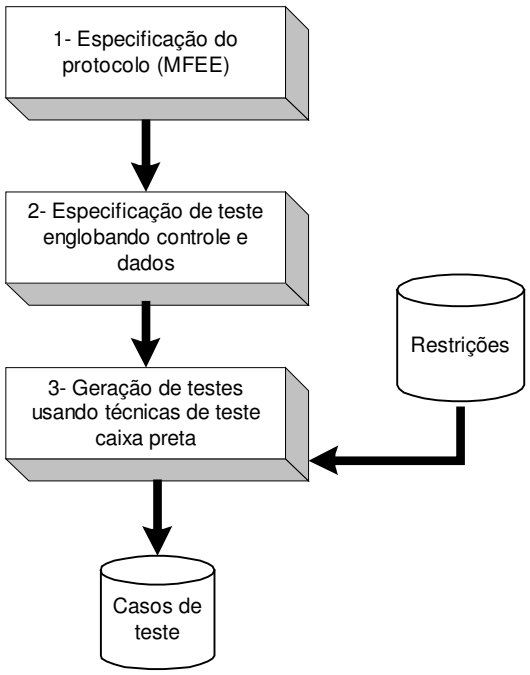

Figura 4.3: Método para geração de casos de teste

Para a geração dos testes, a ConDado requer que a MFEE satisfaça algumas propriedades [Sabiao, 1998]:

- **As transições devem estar na forma normal:** ou seja, não deve conter instruções que influenciem o fluxo de controle. Por exemplo, se uma transição contém um comando *if* ela deve ser substituída por 2 transições: uma que represente o predicado falso e outra o verdadeiro.
- **A máquina deve ser modelada de acordo com modelo de Mealy:** em um determinado momento a máquina só pode estar em um único estado dentre um conjunto finito de estados possíveis. Em resposta a uma entrada, ela pode mudar de estado e produzir uma saída. O estado novo e a saída produzida são dados em função do estado atual e da entrada fornecida.
- **A máquina deve ser mínima:** isto é, a máquina de estados não pode ter estados equivalentes. Dois estados *i* e *j* são ditos equivalentes quando toda seqüência de entrada que começa em *i* ou *j* produz a mesma seqüência de saída.
- **A máquina deve ser inicialmente conexa:** ou seja, todo estado deve ser alcançável a partir do estado inicial.

4.3.1 Geração de testes usando técnicas de teste caixa preta

As técnicas de teste caixa preta usadas para a geração dos casos de teste são [Sabiao, 1998]:

- **Teste de transição de estados:** Os testes gerados pela ConDado devem cobrir todas as transições da MFEE pelo menos uma vez. Por *default*, são considerados caminhos livres de laços, isto é, cada laço é executado apenas uma vez. Entretanto, ela permite geração seletiva e testes de laços. Na geração seletiva são especificadas as transições que se deseja cobrir. Nos testes de laços devem ser especificadas as transições que compõem o laço e o número de vezes que esse laço será executado.
- **Teste de sintaxe:** Usado para gerar casos de teste para as condições válidas dos dados de entrada. Entre as falhas que podem ser encontradas usando o teste de sintaxe estão: comandos válidos que não são aceitos e comandos válidos interpretados de forma incorreta.
- **Teste de partição de equivalência:** Usado para escolher valores aleatórios dentro do domínio definido para cada parâmetro.

4.3.2 Limitações

A ConDado possui algumas limitações. As tratadas pela STER são:

- **Geração de um grande número de casos de teste:** a ConDado gera testes para todas as combinações possíveis das transições especificadas. Isso pode acarretar em um grande número de casos de testes gerados, mesmo usando a geração seletiva de testes.

  Visando mitigar esse problema a STER propõe uma outra forma de reduzir a especificação do sistema: usar análise de riscos, o que permite continuar gerando casos de teste para as funções mais importantes do sistema.

- **Geração de testes não executáveis:** Alguns caminhos da MFEE podem não ser alcançáveis durante a execução dos testes devido às condições de guarda das transições. Os casos de teste que cobrem esses caminhos são chamados de casos de teste não executáveis.

  Para reduzir esse problema a STER usa a parametrização dos dados. Isso garante que os casos de teste gerados satisfarão as condições de guarda.

- **Máquina finita de estados estendida descritas em uma linguagem própria:** A ConDado necessita que a MFEE seja traduzida para uma linguagem própria para que os casos de teste possam ser gerados.

  Para auxiliar nessa tarefa foi usada uma ferramenta de apoio, o Modelador de Máquina de Estados, que traduz a MFEE gráfica para a linguagem usada pela ConDado.

4.4 Considerações finais

Nesse capítulo foram mostradas as ferramentas que apoiaram a STER na geração automatizada de casos de testes. São elas:

- **MME** [Melo, 2003]**:** permite que as máquinas de estados sejam modeladas de forma gráfica e as transforma em uma base de fatos em Prolog.
- **ConDado** [Sabiao, 1998]**:** usada para a geração automatizada de testes a partir da base de fatos gerada pelo MME.

Além disso, foram apresentadas as limitações da ConDado que são contornadas pela STER e como a STER as contorna.

Capítulo 5

STER: Uma Estratégia de testes para sistemas reativos

Como visto no capítulo 1, alguns sistemas reativos devem ter um alto grau de confiabilidade para evitar que ocorram perdas humanas ou financeiras significativas. Uma forma de garantir a confiabilidade desses sistemas é testá-los exaustivamente. Entretanto, como a execução de testes exaustivos é impraticável, eles devem ser testados com um conjunto de casos de teste que permita detectar o maior número possível de falhas. Pensando nisso, foi desenvolvida a STER.

Essa estratégia tem como objetivo auxiliar nos testes dos sistemas reativos gerando casos de teste a partir da especificação formal do mesmo e fornecendo um conjunto de casos de teste que permita encontrar as falhas que representam maior risco para o sistema mesmo quando há restrições de tempo, recursos ou pessoal.

Ela permite que os casos de teste sejam gerados de forma automatizada e leva em consideração as necessidades da indústria descritas no capítulo 1. A STER é composta de duas atividades principais: (i) análise de riscos e (ii) geração de casos de teste. Suas principais características são:

- **É aplicada separadamente para cada um dos atores do sistema:** É necessário gerar um conjunto de casos de teste diferente para cada ator do sistema porque cada um deles "enxerga" o sistema de uma maneira diferente, ou seja, para um determinado ator uma função pode ser mais importante que para outro. O teste por ator permite que a análise de riscos e o conjunto de casos de teste gerados, que é diferente para cada um dos atores, estejam de acordo com as suas necessidades.

- **É baseada em análise de riscos:** A análise de riscos permite encontrar as partes[6] mais importantes do sistema o que possibilita testá-las com mais atenção. Além disso, a STER leva em consideração o ponto de vista dos atores, o que permite dar mais ênfase nos testes das funções que são mais importantes para eles reduzindo a chance de serem encontrados defeitos quando o sistema estiver sendo executado.
- **É baseada em modelos de especificação:** Para facilitar a geração dos casos de teste e permitir que a STER possa ser usada sistematicamente pela indústria são usados modelos UML que, normalmente, são construídos nas fases de análise e projeto do sistema. São usados os casos de uso e os diagramas de seqüência e atividades que, por serem modelos gráficos, facilitam o entendimento da especificação do sistema.
- **Gera os casos de teste a partir de uma máquina finita de estados estendida:** As MFEEs são usadas para especificar formalmente o sistema evitando assim as ambigüidades e imprecisões da especificação em linguagem natural. Essas máquinas são obtidas a partir dos modelos UML. A MFEE foi escolhida no lugar do *statecharts*, que também faz parte da UML, por ser a entrada usada pela ferramenta de geração automatizada de testes descrita no capítulo 4.
- **Conta com o apoio de um ambiente de testes:** o ambiente de testes ajuda a garantir a qualidade dos sistemas reativos gerando e executando uma maior quantidade de testes em comparação à quantidade que seria executada manualmente. O ambiente usado é o ATIFS, descrito no capítulo 4.

A próxima seção descreve o IMEAD, um sistema adotado como estudo de caso e que será usado como exemplo neste capítulo. Os detalhes desse estudo de caso estão no Apêndice A. A análise de riscos proposta é mostrada na seção 5.2. Na seção 5.3 são mostrados os pré-requisitos necessários para a aplicação da STER. Na seção 5.4 são descritos os procedimentos para a execução da estratégia. E por fim na seção 5.5 são apresentadas as considerações finais.

5.1 Descrição do estudo de caso

O IMEAD (Integração de Tecnologia para Implementação de um Modelo de Educação À Distância) usado como estudo de caso foi desenvolvido na Universidade Federal de Viçosa (UFV) sob a orientação de professores do Departamento de Informática/UFV e de desenvolvedores da Central de Processamento de Dados (CPD)/UFV. Seu objetivo é cadastrar e excluir usuários para cursos de ensino à distância via internet, além de permitir alterar e visualizar os dados cadastrados. Deve também conter um link de acesso, para

[6] Por partes do sistema entende-se funções, componentes ou casos de uso, por exemplo. A partir desse ponto, as partes do sistema serão tratadas por funções, sem perda da generalidade.

professores e alunos, à página do curso que será ministrado à distância. O acesso ao sistema será permitido mediante validação da matrícula e senha digitadas pelo usuário.

Os dados dos usuários serão cadastrados por um administrador do sistema. O administrador tem acesso a todas as funcionalidades do sistema, com exceção dos cursos oferecidos.

Os professores e alunos podem alterar e visualizar apenas os seus dados. Cada professor tem acesso somente ao(s) curso(s) que está ministrando. Ele não tem limite de acesso ao(s) curso(s), ou seja, ele pode entrar no curso quantas vezes quiser. Entretanto, o mesmo professor não pode entrar no sistema mais de uma vez simultaneamente. Antes do início do curso, o professor deve definir o número de vezes que cada aluno poderá entrar no sistema. Apenas o administrador poderá alterar o número de acessos permitidos para um aluno.

Cada aluno pode acessar apenas o(s) curso(s) em que ele está matriculado. Além disso, ele tem um número limitado de acessos. Para evitar que vários alunos façam o curso com apenas uma matrícula o sistema não poderá permitir que o mesmo aluno entre no sistema mais de uma vez simultaneamente.

A descrição dos atores do IMEAD e suas funções e o diagrama de casos de uso do sistema são mostrados a seguir na Tabela 5.1 e na Figura 5.1 respectivamente. As descrições dos casos de uso e atores são detalhadas no Apêndice A.

Tabela 5.1: Descrição dos atores do IMEAD e suas funções

Atores	Descrição	Funções
Administrador	Administrador do sistema. Tem acesso a todas as funcionalidades do sistema com exceção dos cursos oferecidos.	➢ Entrar no sistema ➢ Cadastrar novos usuários ➢ Alterar dados ➢ Visualizar dados ➢ Excluir usuários ➢ Sair do sistema
Professor	Professor do curso à distância.	➢ Entrar no sistema ➢ Alterar dados ➢ Visualizar dados ➢ Entrar no curso ➢ Sair do sistema
Aluno	Aluno do curso à distância.	➢ Entrar no sistema ➢ Alterar dados ➢ Visualizar dados ➢ Entrar no curso ➢ Sair do sistema

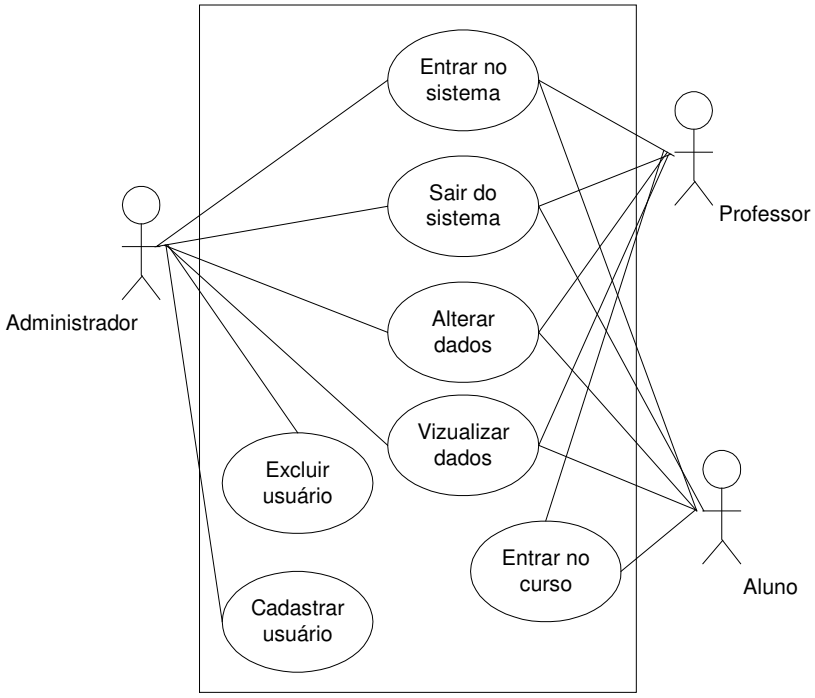

Figura 5.1: Diagrama de casos de uso para o IMEAD

5.2 Análise de riscos

Como visto na seção 2.4, riscos são condições que, se ocorrerem, podem resultar em perdas ou causar algum impacto indesejado. Para garantir a qualidade do sistema é essencial identificar e tratar os riscos associados a ele. Existem riscos que são comuns a todos os sistemas, como por exemplo, o risco de faltar energia elétrica. Entretanto, existem outros que são específicos e dependem de cada sistema. Esses riscos só podem ser identificados por pessoas que conhecem bem o sistema [Pressman, 1997, cap. 6].

A análise de riscos ajuda na identificação e avaliação dos desses riscos. Ela: (i) ajuda determinar quais funções do sistema devem ser testadas com mais cuidado para que o software tenha uma melhor qualidade, (ii) permite que os esforços do teste sejam priorizados e (iii) permite que os recursos disponíveis sejam alocados de forma eficiente. Se os riscos mudarem a análise de riscos deve ser refeita para refletir essas mudanças. Pressman [Pressman, 1997, cap. 6] propõe uma análise de riscos de 4 passos:

1. Identificar os indicadores de riscos em potencial do sistema.
2. Calcular a probabilidade dos riscos ocorrerem (freqüência: alta, média, baixa ou não se aplica).
3. Calcular o impacto dos riscos no sistema (severidade ou peso).
4. Criar uma lista priorizada de funções baseada nos riscos. Onde, priorizar as funções significa ordená-las de acordo com o possível impacto no sistema.

Os casos de teste criados devem levar em consideração a lista priorizada de funções. As funções mais críticas devem ser testadas com mais cuidado que as funções menos críticas. Ou seja, deve ser dada uma maior atenção às funções mais críticas do sistema *sem, no entanto, deixar de lado as outras funções*.

A análise de riscos proposta pela STER consiste em 4 passos como mostrado na Figura 5.2. Esses passos serão detalhados nas subseções a seguir.

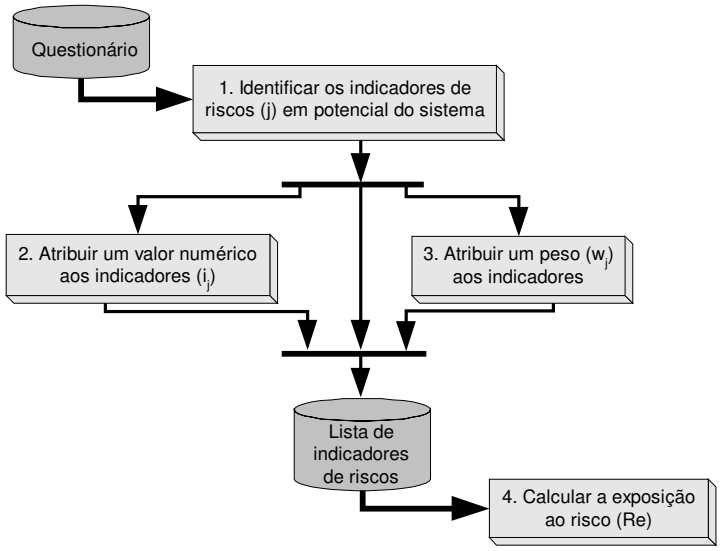

Figura 5.2: Passos da análise de riscos

Essa análise de riscos deve ser feita para cada um dos atores do sistema separadamente, com exceção do *passo1* que deve ser executado para todos os atores em conjunto. Ou seja, primeiro, é criada uma lista de indicadores de riscos em potencial do sistema que *é comum a todos os atores*. Em seguida, de acordo com as necessidades de cada um dos atores, são atribuídos valores numéricos e pesos aos itens que podem ou não ser diferentes para cada um deles.

Passo 1: Identificar os indicadores de riscos em potencial do sistema

Os indicadores de riscos são itens que podem causar algum impacto na qualidade do sistema e por isso devem ser identificados. *Cada sistema deve ter sua própria lista de indicadores*. Eles podem ser, por exemplo, o tamanho de uma função, o número de alterações feitas ou a freqüência relativa de uso, entre outros.

A identificação dos indicadores de riscos em potencial do sistema pode ser feita usando um questionário pré-existente como mostrado por Perry [Perry, 1995, cap.3] ou em reuniões com desenvolvedores, especialistas, gerentes e usuários como proposto por Amland [Amland, 2000]. O ator deve participar dessa identificação, uma vez que é ele quem conhece o sistema como um todo. Esse trabalho em conjunto com o ator auxilia na criação de uma lista com os indicadores de maior relevância para avaliar o sistema.

A STER sugere que a lista de indicadores seja baseada nos questionários fornecido por Perry [Perry, 1995, cap.3]. Mas isso não exclui a possibilidade de que esses indicadores sejam adaptados ou sejam usados outros que não estão nesses questionários. Isto porque eles não são completos e não atendem a todos os tipos de sistema, como por exemplo, os sistemas espaciais. Além disso, a lista final de indicadores deve conter a *freqüência relativa de uso* e a *experiência do ator*. O primeiro identifica a freqüência com que uma função é acionada pelo ator comparada com as outras funções. Ele deve estar presente porque a STER gera os casos de teste levando em consideração o uso que o ator faz do sistema. O segundo representa a experiência que o ator tem na execução do sistema.

Um exemplo de um indicador retirado dos questionários fornecidos por Perry [Perry, 1995, cap. 3] e adaptado para o exemplo IMEAD é mostrado na Tabela 5.2.

Tabela 5.2: Indicador de risco

Indicadores de riscos (j)
5- Grau de confiabilidade na troca de dados com casos de uso externos
• Não envia dados necessários para a operação de outros casos de uso
• Deve enviar/receber dados de/para outro caso de uso
• Deve enviar/receber dados de/para até 50% dos casos de uso
• Deve enviar/receber dados de/para mais de 50% casos de uso

Passo 2: Atribuir um valor numérico aos indicadores

O passo seguinte à identificação dos indicadores de riscos consiste em atribuir um valor numérico a eles. A atribuição desses valores foi baseada na tática de teste proposta por Perry [Perry, 1995, cap. 3] e na a técnica de teste baseada em riscos proposta por Amland [Amland, 2000]. Ambas fornecem valores pré-definidos para o cálculo dos riscos.

Esses valores representam a gravidade do efeito sobre o sistema caso um dos indicadores ocorra. Eles podem ser: risco alto (A) com valor igual a 3, risco médio (M) com valor igual a 2, risco baixo (B) com valor igual a 1 ou não se aplica (N/A) com valor igual a 0. Devem ser atribuídos valores a cada um dos itens do indicador de risco.

Para o indicador da Tabela 5.2, os valores atribuídos aos itens estão mostrados na Tabela 5.3, na coluna *Valores* (i_j)**.**

Tabela 5.3: Atribuição de valores numéricos aos indicadores

Indicadores de riscos (j)	**Valores** (i_j)
5- Grau de confiabilidade na troca de dados com casos de uso externos	
• Não envia dados necessários para a operação de outros casos de uso	N/A =0
• Deve enviar/receber dados de/para outro caso de uso	Baixo = 1
• Deve enviar/receber dados de/para até 50% dos casos de uso	Médio = 2
• Deve enviar/receber dados de/para mais de 50% casos de uso	Alto = 3

Passo 3: Atribuir um peso aos indicadores

O peso de um item indica a importância relativa dele em relação aos outros itens [Perry, 1995, cap. 3]. Ou seja, se o indicador A é mais crítico que o B, então o peso do indicador A deve ser maior que o peso do B. O peso é usado para ponderar as funções que serão testadas e deve ser decidido em conjunto com o usuário.

A classificação dos pesos pode ser feita analisando o grau de criticidade do item de acordo com a Tabela 3.2 (proposta por Mosley [Mosley, 2000, cap. 4]) e atribuindo os valores de acordo com a Tabela 5.4 (atribuição de pesos descrita por Volpi [Volpi, 2001]).

Tabela 5.4: Valores para atribuição dos pesos

Classificação	**Valor** (w_j)
Crítico	5
Prioritário	4
Regular	3
Pouca importância	2
Exceção de qualidade	1

O indicador da Tabela 5.2 foi considerado crítico após analisar seu grau de criticidade usando a Tabela 3.2. Por isso, de acordo com a classificação mostrada na Tabela 5.4, o seu peso é 5 como mostrado na coluna *Peso* (w_j) da Tabela 5.5.

Tabela 5.5: Atribuição do peso aos indicadores

Indicadores de riscos (j)	**Valores** (i_j)	**Peso** (w_j)
5- Grau de confiabilidade na troca de dados com casos de uso externos		5
• Não envia dados necessários para a operação de outros casos de uso	N/A =0	
• Deve enviar/receber dados de/para outro caso de uso	Baixo = 1	
• Deve enviar/receber dados de/para até 50% dos casos de uso	Médio = 2	
• Deve enviar/receber dados de/para mais de 50% casos de uso	Alto = 3	

Passo 4: Calcular a exposição ao risco

O próximo passo é calcular o quanto cada função[7] do sistema está exposta aos riscos. Quanto maior o valor encontrado para a exposição ao risco, maior o risco que o caso de uso representa para o sistema. O valor obtido nesse cálculo será usado para definir quais são os CdUs prioritários do sistema.

Este passo foi baseado na técnica de teste baseada em riscos proposta por Amland [Amland, 2000] e na estratégia de teste proposta por Volpi [Volpi, 2001].

[7] Como a STER trata de casos de uso, a partir desse ponto será calculada a exposição ao risco de cada caso de uso.

Para calcular a exposição ao risco deve-se:

a. Escolher um dos itens dos indicadores de risco para cada caso de uso do sistema: primeiro, deve ser escolhido um item de cada indicador da lista de indicadores para cada um dos CdUs do sistema. Em seguida, deve ser atribuído a ele o valor numérico correspondente ao item escolhido. Uma forma de representar é mostrada na Tabela 5.6.

A Tabela 5.7 mostra a atribuição desses valores para o ator *Administrador* do exemplo IMEAD. Para o indicador *Documentação* do caso de uso *Entrar no Sistema*, foi escolhido o item 1 (*Documentação excelente*) que tem risco baixo (valor 1) que é representado por *B(1)*. Para o caso de uso *Cadastrar usuário*, foi escolhido o item 2 (*Boa documentação, mas com problemas de confiabilidade*), cujo risco é médio (valor 2) e é representado por *M(2)*.

b. Calcular a exposição ao risco de cada CdU: A exposição ao risco ($\mathrm{Re}(f)$) do caso de uso *f* é dada pela fórmula a seguir:

$$\mathrm{Re}(f) = \sum_{j=1}^{n} i_j * w_j$$, onde *i* = [0..3], *w* = [1..5] e *n* é o número total de indicadores

Por exemplo, para o caso de uso *Visualizar Dados* da Tabela 5.7, o cálculo da exposição ao risco ficaria assim:

$$\mathrm{Re}(VisualizarDados) = \sum_{j=1}^{7} i_j * w_j$$

$$\mathrm{Re}(VisualizarDados) = (3\times 2)+(2\times 1)+(2\times 3)+(1\times 2)+(1\times 5)+(2\times 2)+(1\times 1) = 26$$

Tabela 5.6: Modelo para atribuição dos valores numéricos para os indicadores

Caso de uso j	$1(w_1)$	$2(w_2)$	$3(w_3)$	$4(w_4)$	...	$n(w_n)$
Caso de uso 1	i_1	i_2	i_3	i_4	...	i_n
Caso de uso 2	i_1	i_2	i_3	i_4	...	i_n
Caso de uso 3	i_1	i_2	i_3	i_4	...	i_n
...	...	...	...	...	...	...
Caso de uso *m*	i_1	i_2	i_3	i_4	...	i_n

Tabela 5.7: Atribuição dos valores numéricos aos indicadores do IMEAD para o ator *Administrador*

Administrador								
Casos de uso	**Documen-tação (2)**	**Plano de contingência (1)**	**Aprovação do usuário (3)**	**Treinamento do usuário (2)**	**Confiabili-dade (5)**	**Freqüência relativa de uso (2)**	**Experiência do ator (1)**	**Re**
1- Entrar no sistema	B (1)	A (3)	B (1)	B (1)	M (2)	A (3)	B(1)	**27**
2- Cadastrar usuário	M (2)	M (2)	M (2)	A (3)	A (3)	A (3)	M (2)	**41**
3- Excluir usuário	M (2)	M (2)	M (2)	M (2)	A (3)	B (1)	M (2)	**35**
4- Alterar dados	A (3)	M (2)	M (2)	M (2)	M (2)	B (1)	M (2)	**32**
5- Visualizar Dados	A (3)	M (2)	M (2)	B (1)	B (1)	M (2)	B(1)	**26**
6- Sair do sistema	B (1)	N/A (0)	B (1)	B (1)	B (1)	A (3)	B(1)	**19**

5.3 Pré-requisitos

A STER precisa que alguns pré-requisitos sejam satisfeitos para que possa ser usada:

1. **O sistema deve ter sido especificado usando uma extensão para os casos de uso da UML:** Os modelos de especificação usados são os casos de uso da UML, os quais devem ser estendidos como proposto por Binder [Binder 2000, cap. 8] para indicar quais os dados relevantes para os testes. Essa extensão é mostrada na seção 3.1. Ryser et al [Ryser+, 2000] mostra um procedimento que pode ser usado para a criação de casos de uso e cenários. Esse procedimento não será descrito aqui, pois está fora do contexto deste trabalho.

   Os CdUs da UML foram escolhidos porque têm sido amplamente utilizados pela indústria para especificar seus sistemas. Isso permite que a STER possa ser usada por eles de forma sistemática para a geração dos casos de teste.

   Os CdUs estendidos foram usados para levantar informações que não estão presentes nos casos de uso da UML e que são importantes para os testes. As informações usadas são: os parâmetros dos casos de uso, o relacionamento entre as entradas e as saídas, a freqüência relativa de uso e as dependências entre os casos de uso.

2. **Os cenários dos casos de uso devem ser descritos para cada um dos atores:** Um caso de uso descreve um conjunto de cenários para expressar todos os seus detalhes. Em outras palavras, um CdU pode conter vários cenários que o detalham. Um cenário é uma seqüência específica de ações que ilustra o comportamento de um ator mostrando as trocas de mensagens entre ele e o sistema. Os cenários são usados indiretamente na geração automatizada dos casos de testes, uma vez que a MFEE do sistema pode ser criada a partir deles. E essa sim é usada para a geração automatizada dos casos de teste. Uma maneira de representar os cenários está ilustrada na parte inferior da Tabela 5.8 e a descrição dos cenários de um caso de uso do IMEAD é mostrada na Tabela 5.9.

3. **Os atores devem ser descritos:** Cada um dos atores do sistema deve ser descrito de forma individual, uma vez que os casos de testes serão criados para cada um deles separadamente. A freqüência relativa de uso e o nível de habilidade definidos nessa descrição serão usados na análise de riscos definida na seção 5.2. Um modelo para a descrição do ator é mostrado na Tabela 5.10 e um exemplo descrevendo um dos atores do IMEAD é mostrado na Tabela 5.11. A seção 3.4 mostra como descrever os atores do sistema.

Tabela 5.8: Modelo para descrição dos casos de uso e seus cenários

Caso de uso xx (onde xx representa o número do caso de uso) **Nome:** (Nome do caso de uso) **Descrição:** (Breve descrição do caso de uso) **Autores:** (Autores do caso de uso) **Atores:** (Todos os atores do caso de uso) **Prioridade:** (Prioridade do caso de uso – alta, média, baixa) **Pré-condição:** (pré-condições que devem ser estabelecidas antes da execução do caso de uso) **Pós-condição:** (pós-condições que devem ser atingida após a execução do caso de uso) **Parâmetros:** (parâmetros formais de entrada do caso de uso. Deve ter o nome do parâmetro, o tipo e se ele é de: entrada, saída ou entrada/saída)
Ator: (Ator(es) a que se referem os cenários descritos a seguir)
Fluxo normal **Cenário yy** (onde yy representa o número do cenário) **Fluxo alternativo** **Cenário zz** (onde zz representa o número do cenário) **Fluxo de exceção** **Cenário ww** (onde ww representa o número do cenário)

Tabela 5.9: Descrição do caso de uso *Entrar no sistema* e seus cenários

Caso de uso 1 **Nome:** Entrar no sistema **Descrição:** Permite que o usuário entre no sistema. **Autores:** Daniele Constant Guimarães **Atores:** Administrador, professor, aluno. **Prioridade:** A ser calculada **Pré-condição:** O usuário está na página inicial do sistema. **Pós-condição:** A página de com as opções do usuário é exibida corretamente. **Parâmetros:** entrada: senhaUsr: string; entrada/saída: matriculaUsr: string; saída: categoriaUsr: string
Ator: administrador
Fluxo normal: **Cenário 1:** 1. O usuário digita a matrícula e senha corretas. 2. O sistema exibe a página que contém as opções do usuário.

Tabela 5.10: Modelo para descrição dos atores

Ator: (Ator do sistema) **Descrição:** (Breve descrição do ator) **Nível de habilidade:** (experiente, treinado, inexperiente, experiência variada) **Perfil do ator:**	
Caso de uso (lista de todos os casos de uso do sistema)	**Freqüência relativa de uso** (freqüência com que o CdU é acionado comparado com os outros caso de uso: alta, média, baixa, não se aplica (N/A))

Tabela 5.11: Descrição do ator *Administrador*

Ator: Administrador **Descrição:** Gerencia o sistema Imead **Nível de habilidade:** Experiente **Perfil do ator:**	
Caso de uso	**Freqüência relativa de uso**
1– Entrar no sistema	Alta
2– Cadastrar usuário	Alta
3– Excluir usuário	Baixa
4– Alterar dados	Baixa
5– Visualizar Dados	Média
6– Sair do sistema	Alta
7– Entrar no curso	N/A

5.4 Geração dos casos de teste

A Figura 5.3 mostra os principais passos da geração dos casos de teste. Esses passos são explicados nas subseções a seguir. Entretanto, os passos 9 e 10 não serão tratados nesse trabalho. Eles foram deixados para trabalhos futuros por requererem um maior aprofundamento no estudo das técnicas de geração de seqüências de casos de uso e casos de teste.

Vale lembrar que a STER foi desenvolvida para ser aplicada separadamente para cada um dos atores do sistema, pois uma função pode ser mais importante para um ator que para outro. A separação da geração de casos de teste por ator permite que sejam gerados casos de testes diferentes para cada um deles satisfazendo melhor as suas necessidades. Isso quer dizer que, se o sistema tiver 5 atores, serão gerados 5 conjuntos de teste, um para cada ator.

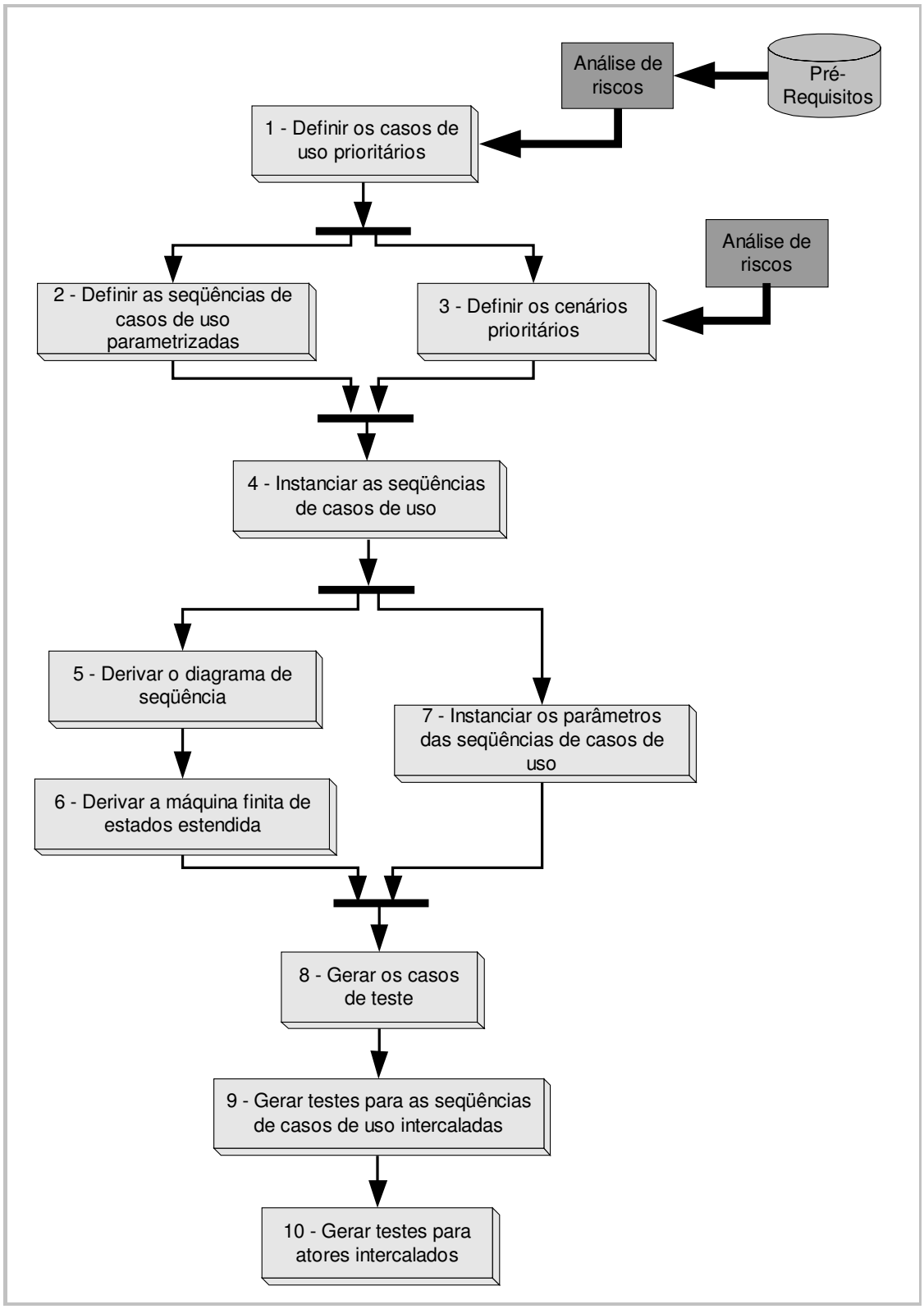

Figura 5.3: Passos para geração dos casos de teste

Passo 1: Definir os casos de uso prioritários

Uma vez que não é possível testar um software exaustivamente, deve-se escolher quais funções do sistema devem ser testadas com mais cuidado. Para que os testes sejam mais eficientes, testando as partes que representam maior risco para o sistema, essa escolha deve ser feita seguindo algum critério de seleção. A STER propõe que seja feita uma análise de riscos, descrita na seção 5.2, para priorizar os casos de uso e a partir daí selecionar a quais deles será data uma maior atenção.

Um exemplo da aplicação da análise de riscos aos CdUs é mostrado a seguir usando o IMEAD. Essa análise foi usada para priorizar os casos de uso do ator *Administrador*.

Primeiro foram identificados os indicadores de riscos. Esses indicadores foram retirados do questionário sugerido por Perry em [Perry, 1995, cap. 3] e adaptados para a aplicação em questão. Foram escolhidos apenas os indicadores que se aplicavam ao exemplo. Os outros foram deixados de lado. Em seguida eles foram quantificados e foi atribuído um peso de cada um deles (passos 1, 2 e 3 da análise de riscos). O resultado desses passos é mostrado na Tabela 5.12. *É importante lembrar que a lista de indicadores mostrada na Tabela 5.12 é só um exemplo e foi elaborada pensando no IMEAD. Cada sistema deve ter sua própria lista de indicadores de acordo com suas características e necessidades.*

O último passo da análise de riscos consiste em calcular a exposição ao risco dos casos de uso usando a fórmula $\mathrm{Re}(f) = \sum_{j=1}^{n} i_j * w_j$. O resultado desse cálculo é mostrado Tabela 5.13. Os detalhes desse cálculo são mostrados no apêndice A.

Para selecionar os casos de uso que serão testados, a STER se baseia no princípio de Pareto, que diz que 80% de todos os defeitos são causados por 20% dos componentes [Pressman, 1997, cap. 16]. Como esses 20% não são conhecidos com antecedência, assumimos que estes componentes são aqueles que participam da realização dos casos de uso de maior risco. Portanto, a STER recomenda que sejam escolhidos pelo menos 20% dos casos de uso do sistema para serem testados. Sendo que devem ser escolhidos os que obtiveram a exposição ao risco mais alta, pois são esses os que representam maior risco para o sistema.

Para o IMEAD foram escolhidos 2 casos de uso. Isto porque existem 6 casos de uso para o ator *Administrador* e 20% deles representam 1.2 CdUs. Os casos de uso escolhidos foram: *Cadastrar Usuário* e *Excluir Usuário*, que obtiveram os maiores *Re*'s, 41 e 35 respectivamente. Eles estão destacados Tabela 5.13.

Tabela 5.12: Lista dos indicadores de risco para o *IMEAD*

Indicadores de riscos (j)	Valores (i_j)	Peso (w_j)
1- Documentação		**2**
• Documentação excelente	Baixo = 1	
• Boa documentação, mas com problemas de confiabilidade	Médio = 2	
• Documentação pobre ou inadequada	Alto = 3	
2- Plano de contingência		**1**
• Não requerido	N/A = 0	
• Plano completo	Baixo = 1	
• Plano existente para maioria das funções	Médio = 2	
• Inexistente	Alto = 3	
3- Aprovação do usuário		**3**
• Aprovação formal do usuário	Baixo = 1	
• Aprovação informal	Médio = 2	
• Em revisão	Alto = 3	
4- Treinamento do usuário		**2**
• Necessário pouco treinamento	Baixo = 1	
• Treinamento formal básico	Médio = 2	
• Treinamento formal avançado	Alto = 3	
5- Grau de confiabilidade na troca de dados com casos de uso externos		**5**
• Não envia dados necessários para a operação de outros casos de uso	N/A =0	
• Deve enviar/receber dados de/para outro caso de uso	Baixo = 1	
• Deve enviar/receber dados de/para até 50% dos casos de uso	Médio = 2	
• Deve enviar/receber dados de/para mais de 50% casos de uso	Alto = 3	
6- Freqüência relativa de uso		**2**
• O ator não usa o caso de uso	N/A =0	
• O ator usa pouco o caso de uso	Baixo = 1	
• O ator usa o caso de uso com uma freqüência média	Médio = 2	
• O ator usa muito o caso de uso	Alto = 3	
7- Experiência do ator		**1**
• O ator não usa o caso de uso	N/A =0	
• O ator é experiente	Baixo = 1	
• O ator é de experiência variada	Médio = 2	
• O ator é inexperiente	Alto = 3	

Tabela 5.13: Exposição ao risco de cada caso de uso

Administrador								
Casos de uso	**Documen-tação (2)**	**Plano de contingência (1)**	**Aprovação do usuário (3)**	**Treinamento do usuário (2)**	**Confiabili-dade (5)**	**Freqüência relativa de uso (2)**	**Experiência do ator (1)**	**Re**
1- Entrar no sistema	B (1)	A (3)	B (1)	B (1)	M (2)	A (3)	B(1)	**27**
2- Cadastrar usuário	M (2)	M (2)	M (2)	A (3)	A (3)	A (3)	M (2)	**41**
3- Excluir usuário	M (2)	M (2)	M (2)	M (2)	A (3)	B (1)	M (2)	**35**
4- Alterar dados	A (3)	M (2)	M (2)	M (2)	M (2)	B (1)	M (2)	**32**
5- Visualizar Dados	A (3)	M (2)	M (2)	B (1)	B (1)	M (2)	B(1)	**26**
6- Sair do sistema	B (1)	N/A (0)	B (1)	B (1)	B (1)	A (3)	B(1)	**19**

Passo 2: Definir as seqüências de casos de uso parametrizadas

Como mostrado nas seções 2.5 e 3.3, os casos de uso são muito úteis para o teste de sistema. Entretanto, eles podem ter restrições e dependências em relação à sua execução. Isso significa que se os casos de uso forem dependentes eles NÃO poderão ser executados em uma ordem arbitrária. Eles deverão ser executados em uma determinada seqüência. Por exemplo, sejam dois casos de uso *x* e *y*. Suponha que *x* seja dependente de *y*, isto é, para que o CdU *x* possa ser executado o CdU *y* deve ter sido executado antes. Então, a ordem de execução dos casos de uso é: *y* seguido de *x*. Nesse caso, dizemos que existe uma seqüência para a execução dos casos de uso[8] que é *y.x*. O ".", como sugerido por Briand et al [Briand+, 2002], é usado para concatenar a seqüência de CdUs.

Como visto na seção 2.5, os casos de uso possuem parâmetros de entrada que determinam o seu comportamento e parâmetros de saída que são o resultado da sua execução. Também podem existir dependências entre esses parâmetros. Isso acontece quando, por exemplo, um parâmetro de saída de um caso de uso é o parâmetro de entrada do próximo CdU da seqüência. A dependência entre os parâmetros mostra o fluxo de dados entre as execuções dos CdUs e por isso são necessários para a geração dos casos de teste [Briand+, 2002].

Se houver dependência entre os CdUs ou entre seus parâmetros ela deve ser levada em consideração na hora de planejar os testes, pois essa dependência pode revelar falhas diferentes daquelas encontradas quando os CdUs são executados isoladamente [Briand+, 2002]. Assim sendo, um dos passos da STER é a definição das seqüências de casos de uso parametrizadas. Para facilitar a identificação e visualização dessa dependência será usado o diagrama de atividades da UML.

Para definir as seqüências de casos de uso parametrizadas é necessário:

a) **Derivar o diagrama de atividades:** No diagrama de atividades que representa a dependência entre os casos de uso, os vértices são os CdUs e as arestas representam as dependências entre eles. Os parâmetros de cada CdU são listados entre parênteses. Esse diagrama deve incluir *todos* os casos de uso para um determinado ator do sistema não só os CdUs prioritários definidos no *passo 1*. Isso é necessário porque um caso de uso prioritário pode precisar de um CdU não prioritário para ser executado. Além disso, o diagrama de atividades deve representar o ciclo de vida de um dado mostrando as operações que podem ser executadas com ele.

   O diagrama de atividades para o exemplo IMEAD é mostrado na Figura 5.4.

[8] A partir desse ponto a seqüência de execução dos casos de uso será tratada como seqüência de casos de uso.

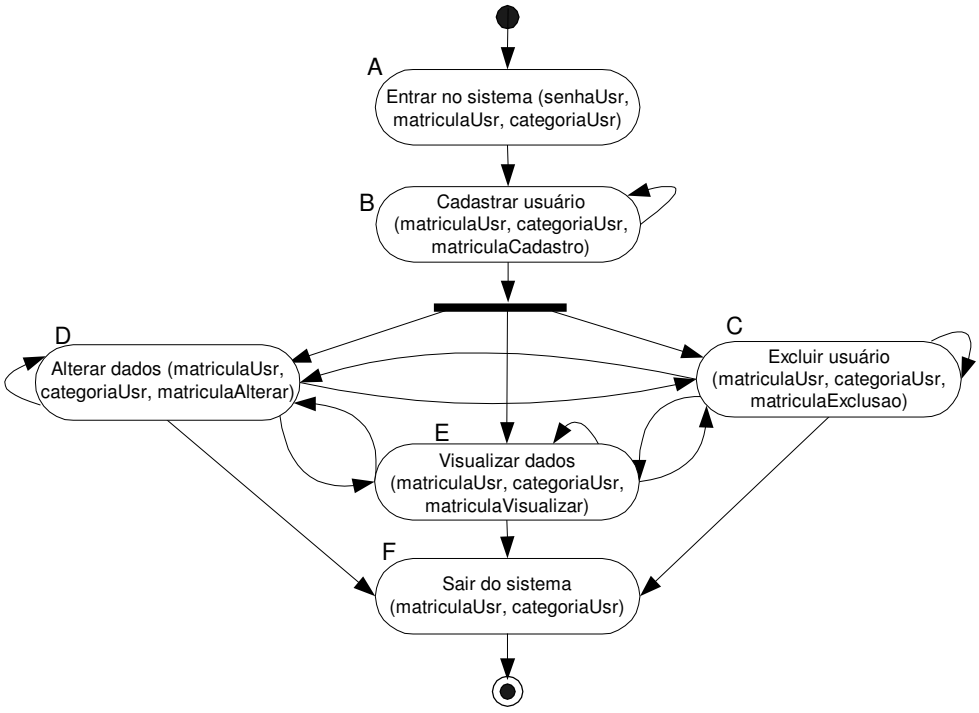

Figura 5.4: Diagrama de atividades do IMEAD

b) Escolher as seqüências de casos de uso que cubram todos os CdUs prioritários: As seqüências de casos de uso são obtidas percorrendo os caminhos no diagrama de atividades. Para percorrer esses caminhos pode ser realizada uma busca em profundidade em um grafo direcionado que seja correspondente ao diagrama de atividades dos casos de uso. No Apêndice A encontra-se o grafo direcionado correspondente ao diagrama de atividades da Figura 5.4 e como ele foi obtido.

Se houver *loops* nesse diagrama, a quantidade de caminhos obtidos pode se tornar infinita. Nesse caso, a STER recomenda que o *loop* seja executado uma única vez, caso não tenha sido especificado o número de vezes que ele deve ser executado.

As seqüências escolhidas devem ser aquelas que contém pelo menos um dos casos de uso prioritário definidos no *passo 1*. Todos os casos de uso prioritários devem ser cobertos.

c) Parametrizar as seqüências de casos de uso: Depois de escolhidas as seqüências, devem ser determinadas as dependências entre os parâmetros dos casos de uso nas seqüências escolhidas. Os parâmetros que tem o mesmo nome e estão na mesma seqüência devem ter o mesmo valor em todos os casos de uso em que são usados. Por exemplo, sejam dois casos de uso *x* e *y* e a seqüência de execução seja *y.x*. Suponha que *x* possua os parâmetros *param1* e *param2* e *y*, os

parâmetros *param1* e *param3*. Para que o fluxo de dados entre *x* e *y* sejam executados, o valor do parâmetro *param1* de *y* deve ser igual ao valor do *param1* de *x*. A representação da seqüência acima seria: *y(param1, param3).x(param1, param2)*. Essas seqüências são chamadas de **seqüências parametrizadas**.

Algumas das seqüências de CdUs possíveis para a Figura 5.4 seriam:

(i) EntrarNoSistema.CadastrarUsuario.ExcluirDados. SairDoSistema

(ii) EntrarNoSistema.CadastrarUsuario.SairDoSistema

(iii) EntrarNoSistema.CadastrarUsuario.AlterarDados.VisualizarDados. AlterarDados.SairDoSistema

No *passo 1* da estratégia foram selecionados os casos de uso *Cadastrar Usuário* e *Excluir Usuário*. Portanto, eles devem estar presentes nas seqüências de CdUs escolhidas. As seqüências escolhidas podem ter apenas um deles ou os dois. O importante é que cada um esteja em pelo menos uma seqüência. Como nesse exemplo todas as seqüências passam obrigatoriamente pelo CdU *Cadastrar Usuário*, poderia ser selecionada a seqüência **(i)** que contém os CdUs *Excluir Usuário* e *Cadastrar Usuário*.

A parametrização das seqüências é feita apenas para aquelas que foram escolhidas. Além disso, são usados os parâmetros dos CdUs definidos durante a criação/extensão dos casos de uso, mostrada na seção 5.3. Os parâmetros dos casos de uso do IMEAD estão definidos no Apêndice A. Assim sendo, a seqüência **(i)** parametrizada seria:

(i) EntrarNoSistema(senhaUsr, matriculaUsr, categoriaUsr).

CadastrarUsuario(matriculaUsr, categoriaUsr, matriculaCadastro).

ExcluirDados(matriculaUsr, categoriaUsr, matriculaExclusao).

SairDoSistema(matriculaUsr, categoriaUsr)

A dependência entre os parâmetros da seqüência de CdUs também deve ser determinada. Para a seqüência **(i)** essa dependência seria:

- a matriculaUsr deve ser a mesma para os CdUs *Entrar No Sistema, Cadastrar Usuário, Excluir Dados* e *Sair Do Sistema*
- a categoriaUsr deve ser a mesma para os CdUs *Entrar No Sistema, Cadastrar Usuário, Excluir Dados* e *Sair Do Sistema*

Passo 3: Definir os cenários prioritários

Como visto na seção 5.3, um cenário é uma seqüência específica de ações que ilustra o comportamento de um ator mostrando as trocas de mensagens entre ele e o sistema. O uso dos cenários garante que os passos importantes para a execução do sistema não serão deixados de lado e que o fluxo de informações está correto [Rumbaugh+, 1991].

Depois de decididos quais são os casos de uso prioritários (*passo 1*) é a vez de definir quais os cenários desses CdUs serão testados. Se o número de cenários de um caso de uso for muito grande pode não ser possível testar todos eles. Além disso, alguns dos cenários podem não ser críticos e assim não seria necessário dar muita ênfase aos seus testes. Por isso, a STER recomenda que também seja feita a análise de riscos descrita na seção 5.2 para priorizar os cenários e determinar quais deles deverão ser testados com mais cuidado. Para a análise de riscos dos cenários a lista dos indicadores com seus valores e pesos devem ser os mesmos usados na análise de riscos dos casos de uso.

Apesar da seqüência de casos de uso possuir CdUs prioritários e não prioritários, para a análise de riscos dos cenários são levados em consideração *apenas os cenários que pertencem aos casos de uso prioritários*. Devem ser avaliados *todos* os cenários de cada um dos CdUs prioritários.

Os detalhes do cálculo da exposição ao risco dos cenários que pertencem aos casos de uso prioritários do ator *Administrador* para o exemplo IMEAD são mostrados no apêndice A e os resultados na Tabela 5.14.

Após a análise de riscos dos cenários, foram selecionados os cenários destacados na Tabela 5.14. São eles: *cenários 2* e *6* do caso de uso *Cadastrar usuário* e *cenários 7* e *12* do caso de uso *Excluir Usuário*. No *cenário 2* um usuário é cadastrado corretamente. No *6*, o administrador não informou todos os dados necessários para o cadastro. No *7*, um usuário é excluído com sucesso e no *12*, o administrador escolheu um usuário incorreto para ser excluído. Essa seleção foi baseada no princípio de Pareto [Pressman, 1997, cap. 16], o mesmo critério usado para a seleção dos casos de uso prioritários apresentado no *passo 1*.

Tabela 5.14: Exposição ao risco de cada cenário que pertence aos casos de uso prioritários

Administrador									
Casos de uso	**Cenários**	**Documen-tação (2)**	**Plano de contingên-cia (1)**	**Aprovação do usuário (3)**	**Treina-mento do usuário (2)**	**Confiabili-dade (5)**	**Freqüência relativa de uso (2)**	**Experiên-cia do ator (1)**	**Re**
2- Cadastrar usuário	**2**	M (2)	M (2)	A (3)	A (3)	A (3)	A (3)	M (2)	**44**
	3	A (3)	A (3)	B (1)	B (1)	A (3)	B (1)	M (2)	**33**
	4	A (3)	N/A (0)	B (1)	B (1)	B (1)	B (1)	M (2)	**20**
	5	A (3)	N/A (0)	B (1)	B (1)	B (1)	B (1)	M (2)	**20**
	6	M (2)	M (2)	A (3)	A (3)	A (3)	M (2)	M (2)	**42**
3- Excluir usuário	**7**	A (3)	M (2)	A (3)	A (3)	A (3)	A (3)	M (2)	**46**
	8	A (3)	A (3)	B (1)	B (1)	A (3)	B (1)	M (2)	**33**
	9	A (3)	N/A (0)	B (1)	B (1)	B (1)	B (1)	M (2)	**20**
	10	A (3)	N/A (0)	B (1)	B (1)	B (1)	B (1)	M (2)	**20**
	11	A (3)	A (3)	A (3)	M (2)	M (2)	M (2)	M (2)	**38**
	12	A (3)	A (3)	A (3)	M (2)	A (3)	M (2)	M (2)	**43**

Passo 4: Instanciar as seqüências de casos de uso

Devem ser criadas instâncias para seqüências de casos de uso escolhidas no *passo 2* de acordo com o número de cenários que devem ser testados e que foram escolhidos no *passo 3*. Se tiver sido selecionado mais de um cenário para um ou mais casos de uso, deve ser criada uma instância do CdU para cada cenário diferente. Se houver apenas um cenário para o caso de uso deverá ser criada apenas uma instância do CdU.

Para os casos de uso que estão nas seqüências selecionadas no *passo 2* mas que não são prioritários, ou seja, que não foram escolhidos no *passo 1*, deve ser criada apenas uma instância. Essa instância corresponde ao cenário que ativa a execução dos casos de uso seguinte.

Por exemplo, seja a seqüência de casos de uso *y(param1, param3).x(param1, param2).* Se o CdU *x* foi escolhido como prioritário e tem 3 cenários: *cx1, cx2* e *cx3*. O CdU *y*, que não foi selecionado como prioritário, tem 2 cenários: *cy1* e *cy2*, onde *cy1* é o cenário que ativa a execução do CdU *x*. Então devem ser criadas 3 instâncias da seqüência, uma para cada cenário de *x*. Assume-se que o único cenário de *y* que vai ser executado é o *cy1*, pois é ele quem ativa a execução de *x*.

Para o IMEAD, será criada apenas uma instância da seqüência de casos de uso **(i)** escolhida no *passo 2*, uma vez que ela pode ser usada para testar os 4 cenários escolhidos no *passo 3*. Como foram selecionados dois cenários para os CdUs *Cadastrar Usuário* e *Excluir Dados,* foram criadas 2 instâncias para cada um deles. Essa instância é mostrada a seguir.

(instância 1) EntrarNoSistema(senhaUsr, matriculaUsr, categoriaUsr).

CadastrarUsuario(matriculaUsr, categoriaUsr, matriculaCadastro).

CadastrarUsuario(matriculaUsr, categoriaUsr, matriculaCadastro).

ExcluirDados(matriculaUsr, categoriaUsr, matriculaExclusao).

ExcluirDados(matriculaUsr, categoriaUsr, matriculaExclusao).

SairDoSistema(matriculaUsr, categoriaUsr)

Passo 5: Derivar o diagrama de seqüência

O diagrama de seqüência descreve a interação entre os atores e o sistema mostrando as mensagens (eventos) que podem ser trocadas entre eles e enfatiza a ordenação temporal dessas mensagens. Ele pode ser usado para representar os cenários de forma gráfica, facilitando assim, o seu entendimento.

Na STER ele é usado como um passo intermediário para a derivação da máquina finita de estados estendida. Sua obtenção, a partir dos cenários, é feita modelando cada uma das mensagens do cenário como uma mensagem enviada/recebida no diagrama de

seqüência. Além disso, a STER recomenda que seja criado um diagrama de seqüência para cada uma das seqüências de casos de uso instanciadas no *passo 4*.

Um exemplo da obtenção do diagrama de seqüência derivado a partir do *cenário 2* do CdU *Cadastrar Usuário* descrito na Tabela 5.15 é mostrado na Figura 5.5.

Tabela 5.15: *Cenário 2* do caso de uso *Cadastrar Usuário*

Fluxo normal: **Cenário 2:** 1. O usuário escolhe a opção cadastrar usuários. 2. O sistema exibe a página para cadastrar usuários. 3. O usuário digita os dados completos. 4. O sistema confirma o cadastro. 5. O usuário escolhe a opção voltar para a página de opções. 6. O sistema exibe a página que contém as opções do usuário.

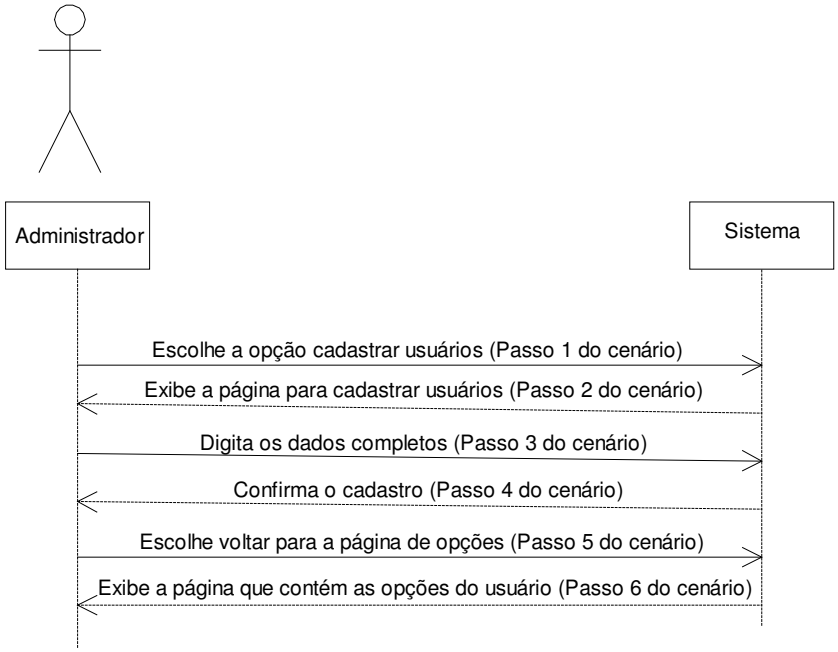

Figura 5.5: Diagrama de seqüência para o *cenário 2* do CdU *Cadastrar Usuário*

Para a (*instância 1)* da seqüência de CdUs do IMEAD definida no *passo 4*, o diagrama de seqüência ficaria como mostrado na Figura 5.6.

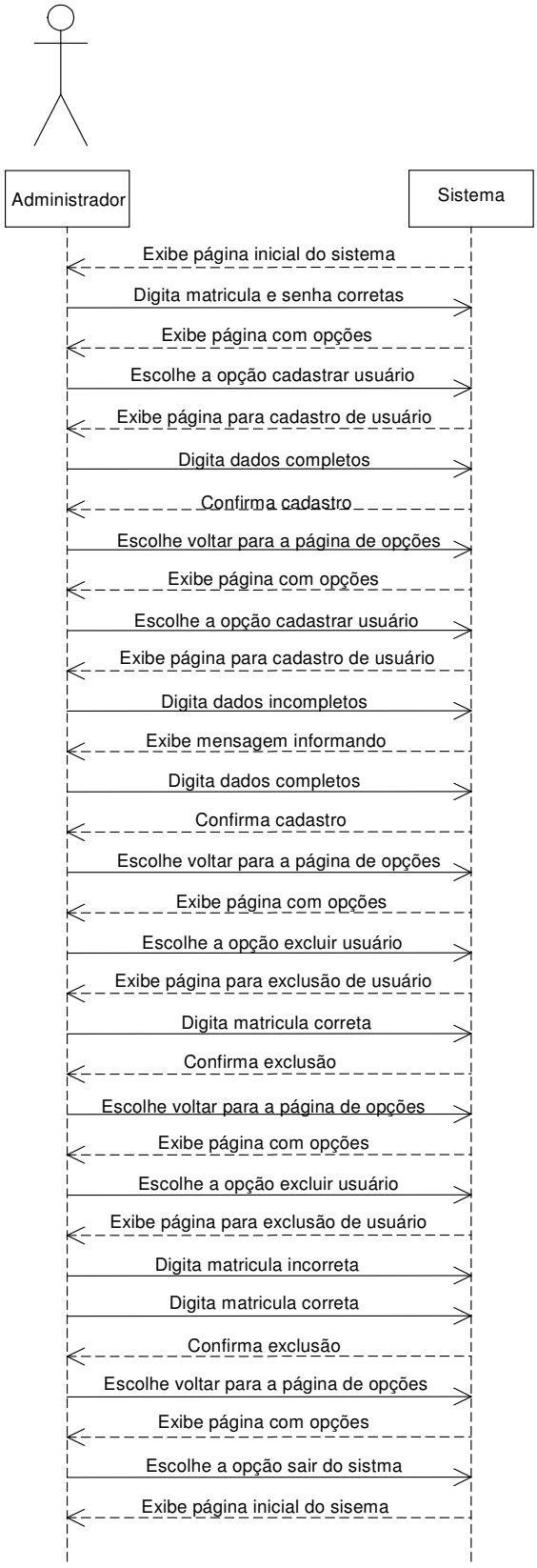

Figura 5.6: Diagrama de seqüência para a *(instância 1)*

Passo 6: Derivar a máquina finita de estados estendida

Rumbaugh et al [Rumbaugh+, 1991, cap. 5] mostra uma forma de derivar a máquina finita de estados clássica (MFE) a partir do diagrama de seqüência. Nela, o intervalo entre dois eventos de entrada no diagrama de seqüência do sistema que está sendo modelado é um estado. Os eventos de entrada são os eventos da MFE. Se o cenário que está sendo modelado puder ser repetido, o caminho na MFE deve ser fechado. Além disso, se uma seqüência de eventos puder ser repetida, deve ser criado um *loop*, onde o estado inicial e o final são os mesmos. Cada cenário do diagrama de seqüência corresponde a um caminho do diagrama de estados.

Um exemplo de derivação da MFE para o diagrama de seqüência da Figura 5.5 seria feita da seguinte forma:

- **Levantar os estados:** os estados foram sublinhados como mostra a Figura 5.7
- **Levantar os eventos de entrada:** os eventos de entrada foram colocados em letras maiúsculas como mostra a Figura 5.7.
- **Construir a MFE:** a MFE derivada do diagrama de seqüência da Figura 5.7 é mostrada na Figura 5.8. Os estados derivados estão sublinhados e os eventos em letras maiúsculas.

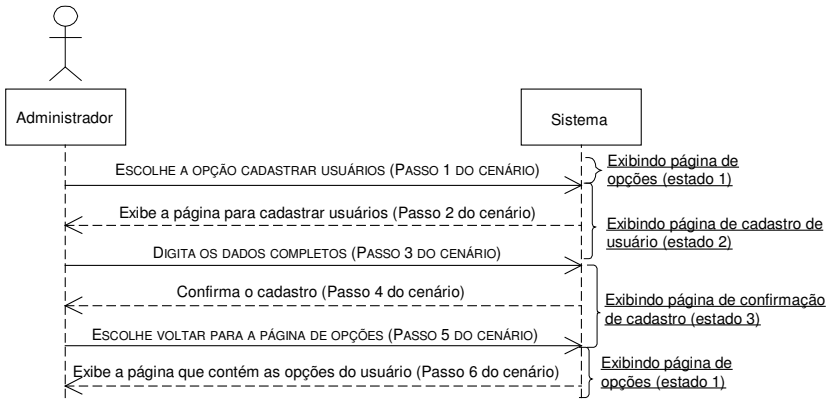

Figura 5.7: Diagrama de seqüência usado na derivação da MFE do *cenário 2* do CdU *Cadastrar Usuário*

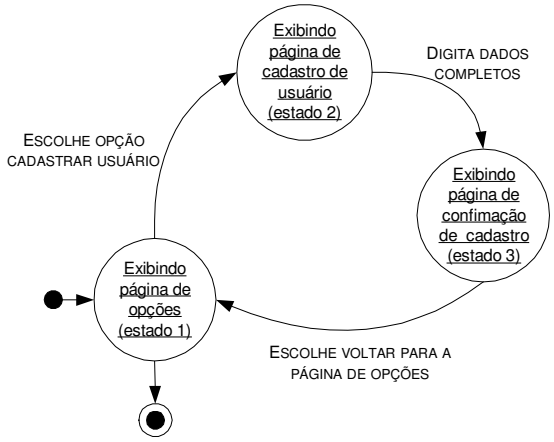

Figura 5.8: MFE para o *cenário 2* do CdU *Cadastrar Usuário*

A STER recomenda que seja criada uma máquina finita de estados estendida a partir de uma máquina finita de estados clássica para cada diagrama de seqüência criado no passo 5. Tanto a MFE quanto a MFEE foram definidas na seção 2.3. A MFEE gerada a partir do diagrama de seqüência da *(instância 1)* mostrado na Figura 5.6 é ilustrado na Figura 5.9.

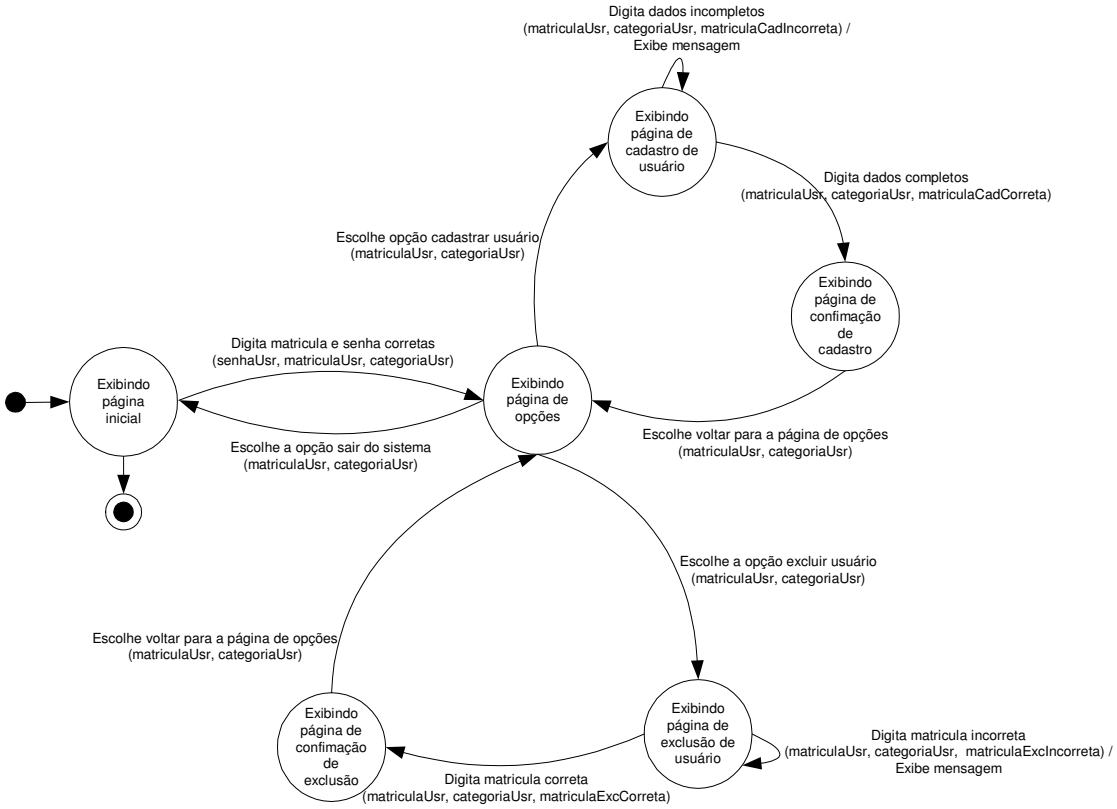

Figura 5.9: MFEE gerada a partir do diagrama de seqüência da Figura 5.6

Passo 7: Instanciar os parâmetros das seqüências de casos de uso

Depois de definidas quantas seqüências de CdUs serão testadas efetivamente (*passo 4*), os parâmetros dos CdUs de cada uma dessas seqüências devem ser instanciados. O testador, em conjunto com o ator, deve determinar o número de instâncias que serão usadas para os parâmetros.

Nessa instanciação são atribuídos valores simbólicos aos parâmetros. Esses valores são usados para a geração automatizada dos casos de teste. Entretanto, para a execução dos testes cada um deles deve ser substituído por um valor real e distinto do domínio de cada parâmetro. Por exemplo, se um parâmetro *parametro1* tiver 2 instâncias, os valores simbólicos poderiam ser *p1* e *p2* e cada um deles deve ser substituído por um valor real e distinto na hora da execução.

Se, para a mesma seqüência de CdUs, for necessário executar mais de um teste com valores diferentes para os parâmetros deve ser criada mais de uma instância para o

parâmetro. E, se houver mais de uma instância para o mesmo parâmetro pode ser gerada uma seqüência de CdUs diferente para cada uma delas, caso seja necessário. Isso pode levar à geração de um grande número de seqüências de CdUs.

A solução para esse problema pode ser a intercalação das seqüências instanciadas (passo 9 da STER), que gera apenas uma seqüência de CdUs com as várias instâncias para o mesmo parâmetro. Entretanto, a geração das seqüências de CdUs intercaladas será deixada para trabalhos futuros por requerer um estudo mais aprofundado sobre o tema.

Um exemplo de instanciação de parâmetros: seja a seqüência de casos de uso *y(param1, param3).x(param1*, *param2)*. Usando 2 instâncias para o *param1* e apenas uma instância para cada um dos outros parâmetros (*param2* e *param3*), as instâncias simbólicas para eles poderiam ser *p1* e *p1a* para o *param1* e *p2* e *p3* para os parâmetros *param2* e *param3* respectivamente. Logo, são geradas duas seqüências de casos de uso com os parâmetros instanciados: (i) *y(p1, p3).x(p1*, *p2)* e (ii) *y(p1a, p3).x(p1a*, *p2)*. Entretanto, se a seqüência de casos de uso fosse *y(param1, param3).y(param1, param3).x(param1*, *param2)* e as instâncias dos parâmetros acima fossem mantidas, poderia ser gerada apenas uma seqüência de casos de uso com os parâmetros instanciados: (i) *y(p1, p3). y(p1a, p3).x(p1*, *p2).*

A instanciação dos parâmetros da (*instância 1)* da seqüência de CdUs definida no *passo 4* ficaria da seguinte forma:

- senhaUsr: senu1
- matriculaUsr: matu1
- categoriaUsr: catu1
- matriculaCadastro: matCadCor1
- matriculaExclusao: matExcCor, matExcInc

Assim, a seqüência de casos de uso da (*instância 1)* com os parâmetros instanciados seria:

- EntrarNoSistema(senu1, matu1, catu1).

  CadastrarUsuario(matu1, catu1, matCadCor1).

  CadastrarUsuario(matu1, catu1, matCadCor1).

  ExcluirDados(matu1, catu1, matExcInc).

  ExcluirDados(matu1, catu1, matExcCor).

  SairDoSistema(matu1, catu1)

Passo 8: Gerar os casos de teste

A geração automatizada dos casos de teste é feita usando as ferramentas ConDado (Controle e Dados) e MME (Modelador de Máquina de Estados), ambas descritas no capítulo 4. Isso é feito entrando com a máquina finita de estados estendida definida no *passo 6* e as instâncias dos parâmetros definidas no *passo 7* no MME, que gera um arquivo com a máquina finita de estados estendida convertida para o formato de entrada da ConDado. De posse desse arquivo, basta executar a ConDado que os casos de teste serão gerados automaticamente.

Os casos de teste gerados para o exemplo IMEAD encontram-se no Apêndice A.

5.5 Considerações finais

Nesse capítulo foram descritos os passos da STER, os pré-requisitos para sua utilização e a análise de riscos adotada. A STER foi criada para auxiliar nos testes de sistemas reativos gerando casos de teste de forma automatizada a partir da especificação formal do sistema para as funções mais críticas do sistema do ponto de vista do ator.

A STER deve ser aplicada separadamente para cada um dos atores do sistema gerando um conjunto de casos de teste diferente para cada um deles. Além disso, ela é baseada em (i) uma análise de riscos para encontrar as funções mais críticas do sistema, (ii) modelos UML e (iii) na máquina finita de estados estendida para facilitar a criação dos casos de teste. Para a geração automatizada dos casos de teste a STER conta com o apoio do ATIFS (Ambiente de Testes baseado em Injeção de Falhas por Software), descrito no capítulo 4.

A análise de riscos adotada pela STER foi baseada (i) na estratégia de teste proposta por Volpi [Volpi, 2001], descrita na seção 3.2 (ii) na técnica de teste baseada em riscos proposta por Amland [Amland, 2000], descrita na seção 3.5, (iii) no processo de seleção de casos de teste baseado nas prioridades do usuário proposto por McGregor et al [McGregor+, 2000], descrito na seção 3.4, (iv) no processo de teste proposto por Mosley [Mosley, 2000, cap. 4], descrito na seção 3.6, (v) na análise de riscos proposta por Pressman [Pressman, 1997, cap. 6], citada na seção 5.2 e (vi) na tática de teste proposta por Perry [Perry, 1995, cap. 3], descrito na seção 3.7.

A utilização de casos de uso e cenários para os testes foi inspirada no método SCENT [Ryser+, 2000], descrito na seção 3.1 e na metodologia TOTEM [Briand+, 2002], descrita na seção 3.3. Além disso, o método SCENT também inspirou a formalização dos casos de uso em máquinas de estado e a metodologia TOTEM inspirou a criação das seqüências de casos de uso parametrizadas e instanciadas.

Capítulo 6

Aplicação da STER a um sistema real

Este capítulo mostra a aplicação da STER a um estudo de caso real. A STER foi usada para desenvolver casos de teste para o software de preparação de plano de vôo do SACI (Satélite de Aplicações CIentíficas), aplicação esta, que foi desenvolvida pelo Instituto Nacional de Pesquisas Espaciais (INPE). As informações necessárias para esse estudo foram obtidas junto à criadora do sistema, que avaliou o resultado final obtido.

Esse software tem como objetivo apoiar o investigador científico na operação remota, via internet, de seus experimentos que estão no satélite. Essa operação permite que sejam enviadas seqüências de telecomandos para o satélite escolhido usando um navegador *web*. Maiores informações sobre essa aplicação podem ser obtidas em [Pereira Junior+, 2000]. A Figura 6. 1 mostra o diagrama de casos de uso para essa aplicação.

A seção 6.1 mostra os pré-requisitos para o uso da STER, na seção 6.2 é mostrada a geração dos casos de teste usando a STER e a seção 6.3 contém as considerações finais do capítulo.

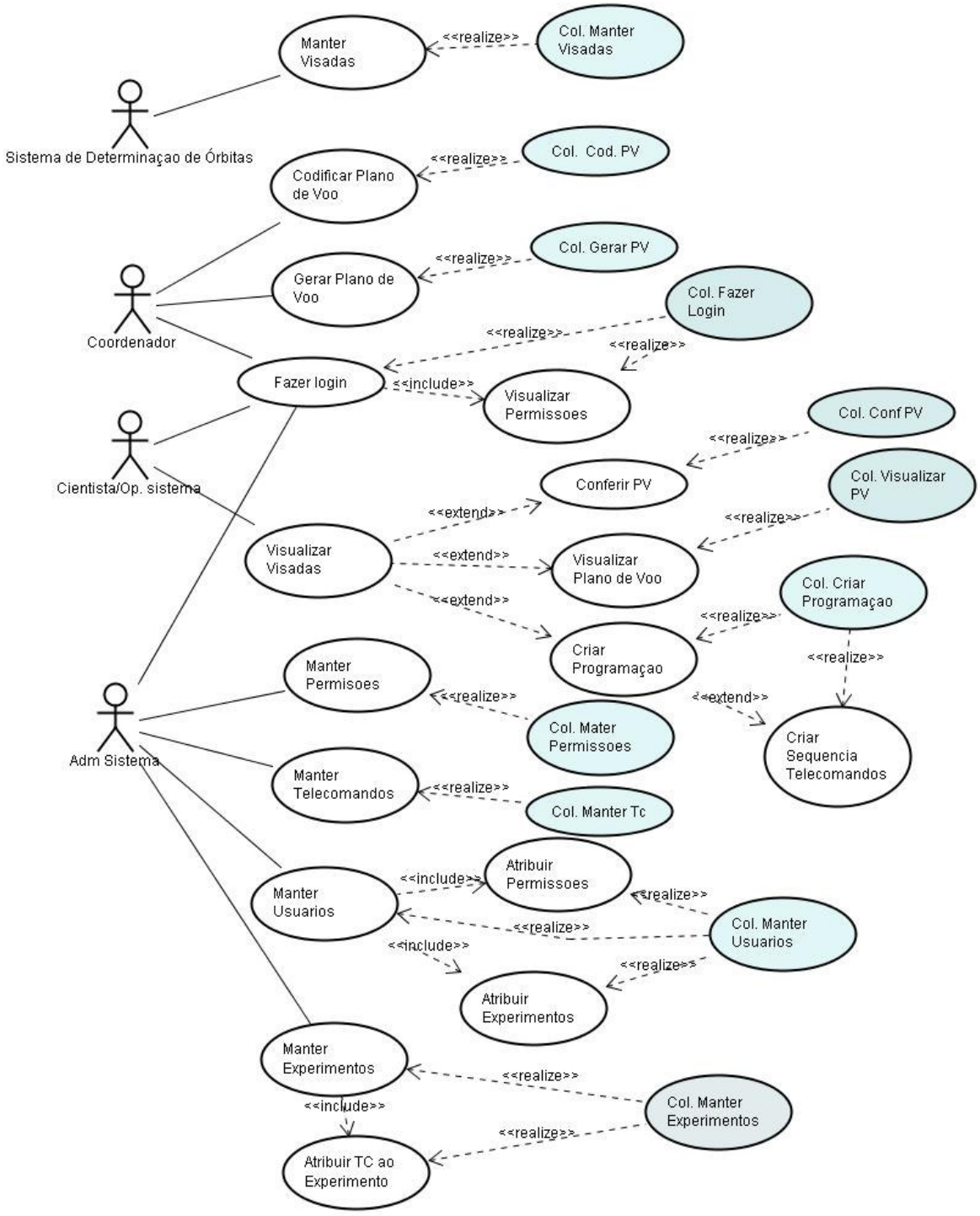

Figura 6. 1: Diagrama de casos de uso do Software de preparação de plano de vôo do SACI

6.1 Pré-requisitos

Nesta seção são descritos os pré-requisitos necessários para o uso da STER. As tabelas de 6.1 a 6.4 descrevem os quatro atores do sistema: Cientista, Operador de Subsistema, Coordenador e Administrador, respectivamente.

Tabela 6.1: Descrição do ator *Cientista*

Ator: Cientista **Descrição:** Responsável pela programação dos telecomandos (TC) a serem enviados para a carga útil sob sua responsabilidade **Nível de habilidade:** Experiente **Perfil do ator:**	
Caso de uso	**Freqüência relativa de uso**
1- Fazer *Login*	Alta
2- Visualizar Permissões	Alta
3- Visualizar Visada	Alta
4- Criar Programação	Alta
5- Visualizar Plano de Vôo	Média
6- Conferir Plano de Vôo	Média

Tabela 6.2: Descrição do ator *Operador de Subsistema*

Ator: Operador de Subsistema **Descrição:** Responsável pela programação dos telecomandos a serem enviados para o subsistema sob sua responsabilidade **Nível de habilidade:** Experiente **Perfil do ator:**	
Caso de uso	**Freqüência relativa de uso**
1- Fazer *Login*	Alta
2- Visualizar Permissões	Alta
3- Visualizar Visada	Alta
4- Criar Programação	Alta
5- Visualizar Plano de Vôo	Média
6- Conferir Plano de Vôo	Média

Tabela 6.3: Descrição do ator *Coordenador*

Ator: Coordenador **Descrição:** Responsável por compatibilizar as programações de telecomandos feitas pelos cientistas e operadores para cada visada do satélite **Nível de habilidade:** Experiente **Perfil do ator:**	
Caso de uso	**Freqüência relativa de uso**
1 – Fazer *Login*	Alta
2- Visualizar Permissões	Alta
3- Visualizar Visada	Alta
4- Gerar Plano de Vôo	Baixa
5- Codificar Plano de Vôo	Média
6- Visualizar Plano de Vôo	Alta

Tabela 6.4: Descrição do ator *Administrador*

Ator: Administrador **Descrição:** Responsável pela base de dados e infraestrutura operacional da rede e Centro de Controle da Missão **Nível de habilidade:** Experiente **Perfil do ator:**	
Caso de uso	**Freqüência relativa de uso**
1- Fazer *Login*	Média
2- Visualizar Permissões	Média
3- Manter Permissões	Baixa
4- Manter Telecomando	Baixa
5- Manter Usuários	Baixa
6- Atribuir Experimento	Baixa
7- Atribuir TC ao experimento	Baixa

As tabelas de 6.5 a 6.9 descrevem os casos de uso estendidos e os cenários para os quais serão criados casos de teste.

Tabela 6.5: Descrição do caso de uso *Fazer login*

Caso de uso 1 **Nome:** Fazer *login* **Descrição:** Provê acesso ao sistema, através de *login* e senha, para que o perfil de operação do usuário seja identificado e conseqüentemente autorizado. **Autores:** ... **Atores:** Cientista, operador do sistema, coordenador de operação, administrador do sistema. **Pré-condição:** Os usuários autorizados devem estar cadastrados na base de dados do sistema. **Pós-condição:** **Parâmetros:** entrada: *login*, senha, satélite; saída: id
Atores: Cientista, operador do sistema, coordenador de operação, administrador do sistema
Fluxo normal: **Cenário 1:** 1. Usuário acessa o *site* do sistema. 2. Sistema apresenta janela com campos para inserção de *login*/senha e escolha do satélite. 3. Usuário digita os dados solicitados. 4. Sistema disponibiliza as funcionalidades associadas ao perfil do usuário.

Tabela 6.6: Descrição do caso de uso *Visualizar permissões*

Caso de uso 2 **Nome:** Visualizar permissões **Descrição:** Apresenta ao usuário uma janela com as funções do sistema autorizadas ao perfil de operador. **Autores:** ... **Atores:** Cientista, operador do sistema, coordenador de operação, administrador do sistema. **Pré-condição:** Os perfis dos usuários devem estar cadastrados na base de dados do sistema. **Pós-condição:** As funcionalidades associadas ao perfil do usuário devem ser disponibilizadas. **Parâmetros:** entrada/saída: id
Atores: Cientista, operador do sistema, coordenador de operação, administrador do sistema
Fluxo normal: **Cenário 2:** 1. O sistema busca na base de dados as permissões associadas ao id do usuário. 2. O sistema apresenta a janela com as funcionalidades de operação permitidas.

Tabela 6.7 Descrição do caso de uso *Visualizar visada*

Caso de uso 3 **Nome:** Visualizar visada **Descrição:** Apresenta a lista das próximas visadas do satélite geradas pelo sistema de determinação de órbita para que uma delas seja selecionada. **Autores:** ... **Atores:** Cientista, operador do sistema, coordenador de operação. **Pré-condição:** Usuário *logado* no sistema. **Pós-condição:** Exibe uma lista com as próximas visadas apresentada na tela do usuário. **Parâmetros:** entrada/saída: id
Atores: Cientista, operador do sistema, coordenador de operação
Fluxo normal: **Cenário 3:** 1. Após reconhecimento do *login*/senha do usuário o sistema apresenta a lista das próximas visadas na tela.

Tabela 6.8: Descrição do caso de uso *Criar programação*

Caso de uso 4 **Nome:** Criar programação **Descrição:** O operador de Subsistema/Cientista autorizado monta as seqüências de telecomandos que compõem a programação a ser transmitida para seu subsistema em uma dada visada. **Autores:** ... **Atores:** Cientista, operador do sistema. **Pré-condição:** Os telecomandos apresentados para o operador/cientista devem estar de acordo com o perfil do usuário. O operador do subsistema deve estar *logado* no sistema e a visada deve ter sido selecionada. **Pós-condição:** A programação deve ser criada e gravada na base de dados **Parâmetros:** entrada: TC; entrada/saída: id, visada, experimento; saída: seqüência de TC, posição do TC
Atores: Cientista, operador do sistema.
Fluxo normal: **Cenário 4:** 1. Operador/cientista escolhe a opção criar programação. 2. Sistema exibe a página para criar programação. 3. Operador/cientista seleciona a visada, o experimento que deseja programar, o TC e a seqüência e a posição do TC na seqüência. **Fluxo de exceção:** **Cenário 5:** 3a. Lista de TCs incompatível com o perfil do usuário.

Tabela 6.9: Descrição do caso de uso *Conferir plano de vôo*

Caso de uso 5 **Nome:** Conferir plano de vôo **Descrição:** O plano de vôo consolidado pelo coordenador de operação é conferido pelos cientistas e operadores de subsistemas. **Autores:** ... **Atores:** Cientista, operador do sistema. **Pré-condição:** O plano de vôo deve ter sido criado pelo coordenador. **Pós-condição:** O plano de vôo deve estar disponível e o aceite gravado na base de dados. **Parâmetros:** entrada/saída: id, visada, plano de vôo
Atores: Cientista, operador do sistema.
Fluxo normal: **Cenário 6:** 1. O sistema envia uma mensagem para o operador/cientista revisar o plano de vôo. 2. O operador/cientista seleciona a visada. 3. O operador/cientista solicita revisão. 4. O sistema retorna o plano de vôo relativo à visada solicitada para que seja revisado. 5. O operador/cientista aceita ou rejeita o plano de vôo. **Fluxo de exceção:** **Cenário 7:** 4a. Plano de vôo não disponível.

6.2 Geração dos casos de teste usando a STER

Nesta seção são mostrados apenas os resultados da aplicação da STER para os atores cientista/operador de sistema. Para maiores informações sobre os passos da STER o capítulo 5 deve ser consultado.

Passo 1: Definir quais os casos de uso prioritários

A análise de riscos mostrada na seção 5.2 foi usada para determinar quais são os casos de uso prioritários do sistema.

Os indicadores de riscos foram selecionados pela criadora do sistema tendo como base o questionário proposto por Perry [Perry, 1995, cap. 3] e a aplicação em questão. A lista dos indicadores de riscos identificados é mostrada na Tabela 6.10. A Tabela 6.11 mostra o resultado do cálculo da exposição ao risco dos casos de uso. Com base nos resultados mostrados na Tabela 6.11 os casos de uso escolhidos foram: *Criar Programação* e *Conferir Plano de Vôo*, que obtiveram os maiores *Re*'s, 90 e 88 respectivamente.

Tabela 6.10: Lista dos indicadores de risco

Indicadores de riscos (j)	Valores (i_j)	Peso (w_j)
1- Necessidade de plano de contingência em caso de falha da função		**5**
Dispensável	Baixo = 1	
Desejável	Médio = 2	
Essencial	Alto = 3	
2- Importância do relacionamento entre atores		**4**
Não requer interação	Baixo = 1	
Pouca interação	Médio = 2	
Atividades complementares	Alto = 3	
3- Experiência exigida do usuário na operação		**5**
Dispensável	Baixo = 1	
Desejável	Médio = 2	
Crítica	Alto = 3	
4- Dependência do processamento		**1**
Dispensável	Baixo = 1	
Desejável	Médio = 2	
Essencial	Alto = 3	
5- Freqüência relativa de uso		**3**
Baixa	Baixo = 1	
Média	Médio = 2	
Alta	Alto = 3	
6- Dependência de sistemas externos		**1**
Não interage com outros sistemas	Baixo = 1	
Interage com poucos sistemas externos	Médio = 2	
Interage com múltiplos sistemas	Alto = 3	
7- Confiabilidade requerida das entradas		**5**
Cheque detalhado de limites	Baixo = 1	
Cheque de consistência básico	Médio = 2	
Não faz validação das entradas	Alto = 3	
8- Status da documentação		**4**
Completa e atualizada	Baixo = 1	
Mais que 75% completa e atualizada	Médio = 2	
Não existente ou desatualizada	Alto = 3	
9- Disponibilidade de facilidades de testes específicas para o teste de subsistema		**4**
Completa ou não requerida	Baixo = 1	
Limitada	Médio = 2	
Não disponível	Alto = 3	
10- Complexidade do sistema projetado		**2**
Única função	Baixo = 1	
Múltiplas funções, mas relacionadas	Médio = 2	
Funções críticas	Alto = 3	
11- Experiência do ator		**5**
O ator não usa o caso de uso	N/A =0	
O ator é experiente	Baixo = 1	
O ator é de experiência variada	Médio = 2	
O ator é inexperiente	Alto = 3	

Tabela 6.11: Exposição ao risco de cada caso de uso

Cientista/ Operador de Sistema												
Casos de Uso	**1 (5)**	**2 (4)**	**3 (5)**	**4 (1)**	**5 (3)**	**6 (1)**	**7 (5)**	**8 (4)**	**9 (4)**	**10 (2)**	**11 (5)**	**Re**
1 – Fazer Login	A (3)	B (1)	NA (0)	NA (0)	A (3)	B (1)	A (3)	A (3)	M (2)	B (1)	B (1)	**61**
2- Visualizar Permissões	B (1)	B (1)	NA (0)	NA (0)	A (3)	B (1)	NA (0)	M (2)	M (2)	M (2)	B (1)	**44**
3- Visualizar Visada	B (1)	B (1)	M (2)	NA (0)	A (3)	M (2)	NA (0)	M (2)	M (2)	M (2)	B (1)	**55**
4- Criar Programação	A (3)	M (2)	A (3)	B (1)	A (3)	M (2)	A (3)	B (1)	A (3)	M (2)	B (1)	**90**
5- Visualizar Plano de Vôo	A (3)	B (1)	A (3)	B (1)	M (2)	B (1)	NA (0)	M (2)	M (2)	M (2)	B (1)	**67**
6- Conferir Plano de Vôo	A (3)	A (3)	A (3)	B (1)	M (2)	M (2)	M (2)	B (1)	A (3)	A (3)	B (1)	**88**

Passo 2: Definir as seqüências de casos de uso parametrizadas

Os diagramas de atividades mostrados nas Figuras 6.2 e 6.3 foram usados para definir as seqüências de casos de uso.

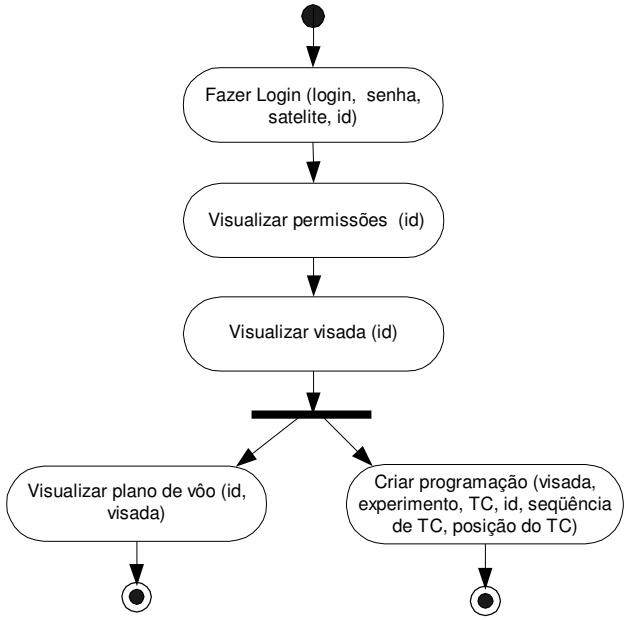

Figura 6.2: Seqüência de casos de uso 1

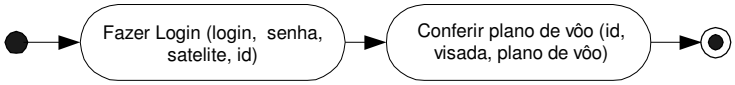

Figura 6.3: Seqüência de casos de uso 2

Como existem dois diagramas de atividades, foram criadas duas seqüências de CdUs:

- *FazerLogin. VisualizarPermissões. VisualizarVisada. CriarProgramação* **(Seqüência 1)**
- *FazerLogin. ConferirPlanoDeVôo* **(Seqüência 2)**

Usando os parâmetros dos CdUs apresentados na seção 6.1, as seqüências de CdUs **(Seqüência 1)** e **(Seqüência 2)** parametrizadas são:

- *FazerLogin (login, senha, satelite, id). VisualizarPermissões (id). VisualizarVisada (id). CriarProgramação (id, visada, experimento, TC, seqüência de TC, posição do TC)* **(Seqüência 1 parametrizada)**
- *FazerLogin (login, senha, satelite, id). ConferirPlanoDeVôo (id, visada, plano de vôo)* **(Seqüência 2 parametrizada)**

Para as seqüências acima, o parâmetro *id* é o mesmo para todos os casos de uso que fazem parte da mesma seqüência.

Passo 3: Definir os cenários prioritários

Usando a Tabela 6.10, foi feita a análise de riscos mostrada na seção 5.2 para identificar quais são os cenários prioritários que pertencem aos casos de uso prioritários. O resultado é mostrado na Tabela 6.12. Com base nesses resultados foram selecionados os cenários de exceções dos casos de uso *Criar Programação* e *Conferir Plano de Vôo*.

Tabela 6.12: Exposição ao risco de cada cenário dos casos de uso prioritários

Cientista/ Operador de Sistema													
Casos de Uso	**Cenários**	**1 (5)**	**2 (4)**	**3 (5)**	**4 (1)**	**5 (3)**	**6 (1)**	**7 (5)**	**8 (4)**	**9 (4)**	**10 (2)**	**11 (5)**	**Re**
Criar Programação	Primário	B (1)	B (1)	B (1)	B (1)	M (2)	A (3)	M (2)	A (3)	B(1)	A(3)	B (1)	**61**
	Exceções	A (3)	B (1)	B (1)	B (1)	B (1)	A (3)	M (2)	A (3)	B(1)	B(1)	B (1)	**64**
Conferir Plano de Vôo	Primário	B (1)	M (2)	M (2)	B (1)	B (1)	A (3)	M (2)	A (3)	B(1)	B(1)	B (1)	**63**
	Exceções	M (2)	M (2)	B (1)	B (1)	B (1)	A (3)	M (2)	A (3)	B(1)	B(1)	B (1)	**65**

Passo 4: Instanciar as seqüências de casos de uso

Com apenas uma instância de cada uma das seqüências de CdUs é possível testar os cenários escolhidos no passo 3. Por isso, foi criada apenas uma instância para cada uma das seqüências parametrizadas definidas no passo 2. Essas instâncias são mostradas abaixo.

- *FazerLogin (login, senha, satelite, id). VisualizarPermissões (id). VisualizarVisada (id). CriarProgramação (id, visada, experimento, TC, seqüência de TC, posição do TC)* **(Seqüência 1 instanciada)**
- *FazerLogin (login, senha, satelite, id). ConferirPlanoDeVôo (id, visada, plano de vôo)* **(Seqüência 2 instanciada)**

Passo 5: Derivar o diagrama de seqüência

As figuras 6.4 e 6.5 mostram o diagrama de seqüência para a **(Seqüência 1 instanciada)** e **(Seqüência 2 instanciada)** definidas no passo 4, respectivamente. Como houve apenas uma instância para cada seqüência de CdUs, então foi criado apenas um diagrama de seqüência para cada uma delas.

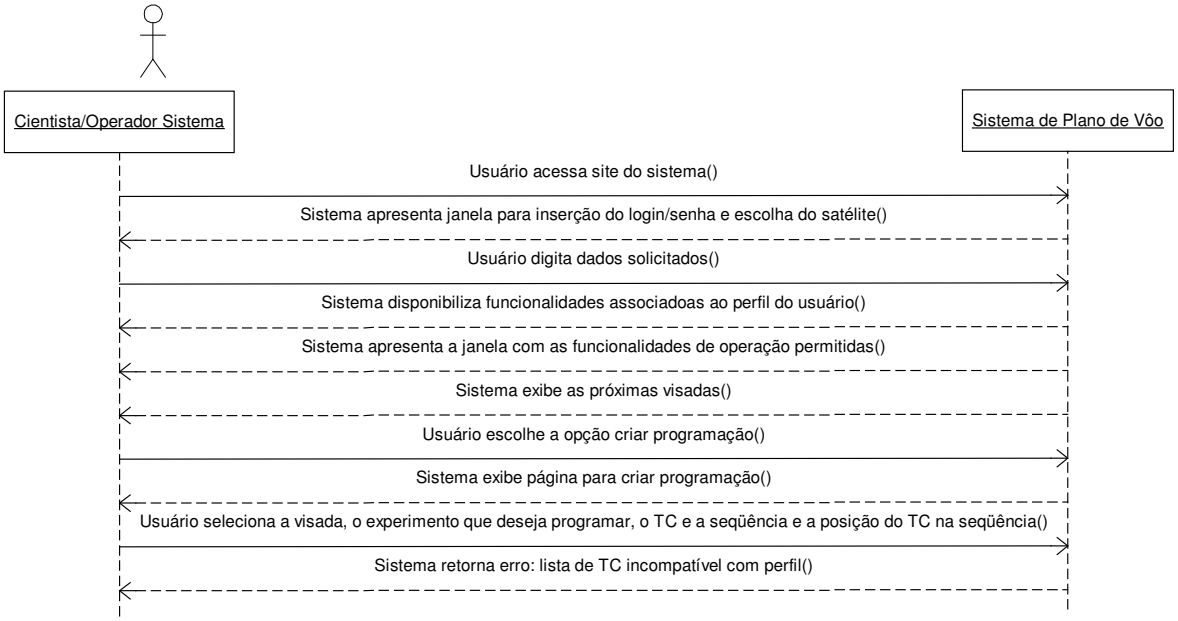

Figura 6.4: Diagrama de seqüência para a *(Seqüência 1 instanciada)*

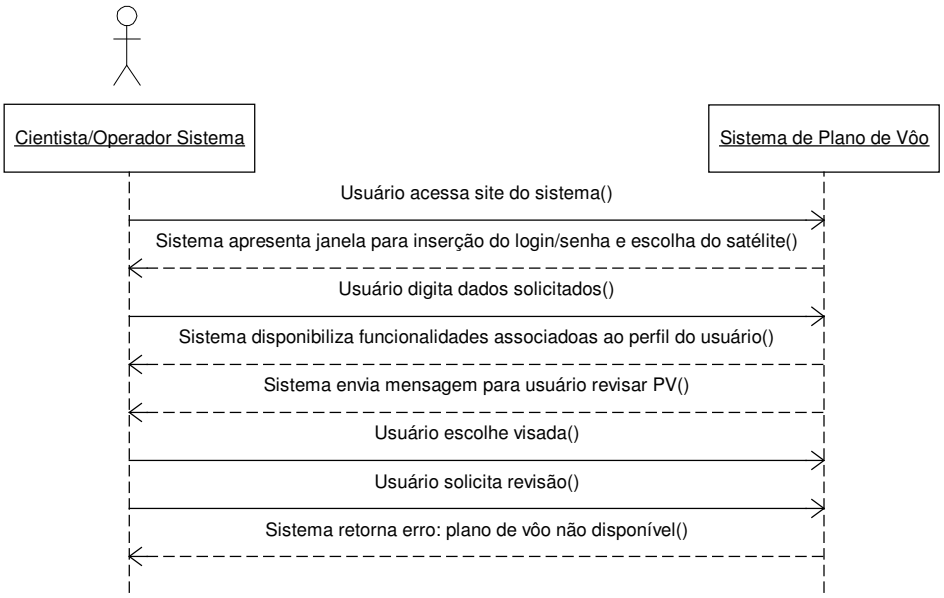

Figura 6.5: Diagrama de seqüência para a *(Seqüência 2 instanciada)*

Passo 6: Derivar a máquina finita de estados estendida

As figuras 6.6 e 6.7 mostram a MFEE criada a partir dos diagramas de seqüência definidos no passo 5, que são mostrados nas figuras 6.4 e 6.5, respectivamente.

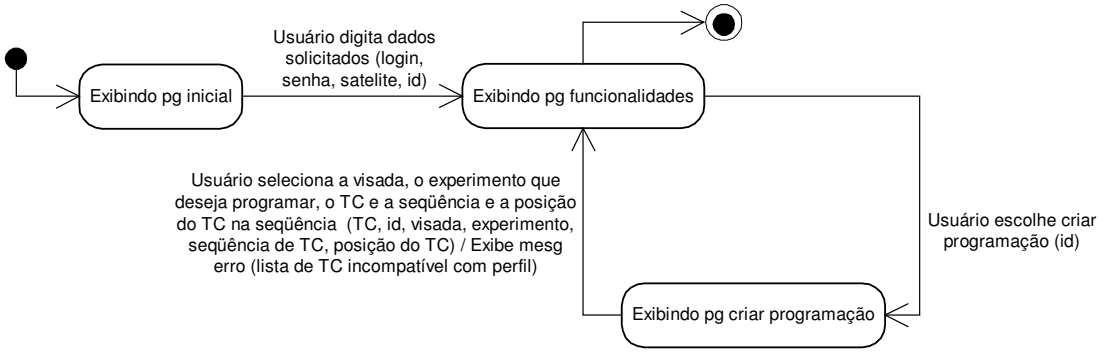

Figura 6.6: MFE para a o diagrama de seqüência da *(Seqüência 1 instanciada)*

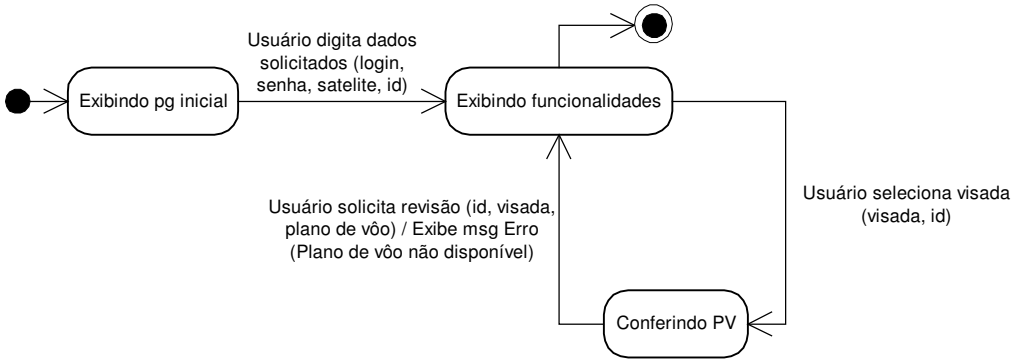

Figura 6.7: MFE para a o diagrama de seqüência da *(Seqüência 2 instanciada)*

Passo 7: Instanciar os parâmetros das seqüências de casos de uso

Foram criadas as seguintes instâncias para os parâmetros das seqüências de casos de uso:

- login: login1, login2
- senha: senha1, senha2
- satelite: satelite1
- id: id1, id2
- visada: visada1
- experimento: experimento1
- TC: tc1
- sequenciaTC: seqtc1
- posicaoTC: postc1
- PlanoVoo: pv1

Assim, as seqüências de casos de uso com parâmetros instanciados seriam:

- *Fazer login (login1, senha1, satelite1, visada1, id1). Visualizar permissões (id1). Visualizar visadas (id1). Criar programação (id1, visada1, experimento1, tc1, seqtc1, postc1)* **(Seqüência 1 com parâmetros instanciados)**
- *Fazer login (login2, senha2, satelite1, visada1, id2). Conferir plano de vôo (id2, visada1, pv1)* **(Seqüência 2 com parâmetros instanciados)**

Passo 8: Gerar os casos de teste

Usando a ConDado (Controle e Dados) e o MME (Modelador de Máquina de Estados), ambas descritas no capítulo 4, foi gerado um caso de teste para a **(Seqüência 1 com parâmetros instanciados)** e um para a **(Seqüência 2 com parâmetros instanciados)**:

- Seqüência 1:
  a. Usuário digita dados solicitados (login1, senha1, satelite1, id1).
  b. Usuário escolhe criar programação (id1).
  c. Usuário seleciona a visada, o experimento que deseja programar, o TC e a seqüência e a posição do TC na seqüência (tc1, id1, visada1, experimento1, seqtc1, postc1)
  d. O sistema retorna uma mensagem de erro dizendo que a lista de telecomandos é incompatível com o perfil do usuário.
- Seqüência 2:
  a. Usuário digita dados solicitados (login2, senha2, satelite1, id2).
  b. Usuário escolhe a visada (visada1, pv1, id2).
  c. Usuário solicita revisão (visada1, id2)
  d. O sistema retorna uma mensagem de erro dizendo que o plano de vôo não está Disponível.

Capítulo 7

Resultados

A STER foi usada em dois estudos de caso: o IMEAD, apresentado no capítulo 5, e o software de preparação de plano de vôo do SACI, apresentado no capítulo 6.

Na próxima seção são mostrados os resultados obtidos na geração de casos de teste para o IMEAD. A seção 7.2 apresenta os resultados obtidos na geração de casos de teste para o software de preparação de plano de vôo do SACI.

7.1 Estudo de caso 1: IMEAD

Foram gerados 5 conjuntos de casos de teste para o IMEAD:

i. **Usando a máquina finita de estados estendida que modela todos os cenários do IMEAD:** Foi gerado um arquivo de casos de teste parcial com aproximadamente de 10Gb de tamanho em disco. Não havia espaço no computador para gerar o arquivo completo.

ii. **Usando a MFEE que não possui parte dos cenários:** Foram retirados todos os cenários de exceção, que foram modelados como *autoloops*, da MFEE acima. Assim como descrito no item 1, foi gerado um arquivo de casos de teste parcial com aproximadamente de 10Gb de tamanho em disco e não havia espaço no computador para gerar o arquivo completo.

iii. **Usando a MFEE que não possui um caso de uso completo:** Foi retirado um caso de uso completo da MFEE descrita no item 1. Foram retirados os 3 estados relativos a esse caso de uso. Os demais estados foram mantidos com todas as suas transições.

iv. **Usando a MFEE que não possui um caso de uso completo nem parte dos cenários:** Foram retirados um caso de uso completo (3 estados) e todos os cenários de exceção da MFEE descrita no item 1.

v. **Usando a STER.**

Os resultados obtidos são mostrados na Tabela 7.1. A Figura 7.1 mostra MFEE que modela todos os cenários do IMEAD.

Tabela 7.1: Resultados IMEAD

Testes	Nº transições	Nº estados	Nº de casos de teste	Tempo de execução aproximado
(i)	39	11	-	4h
(ii)	29	11	-	2h
(iii)	27	8	23486	< 1 minuto
(iv)	20	8	13699	< 1 minuto
(v)	10	5	9	-

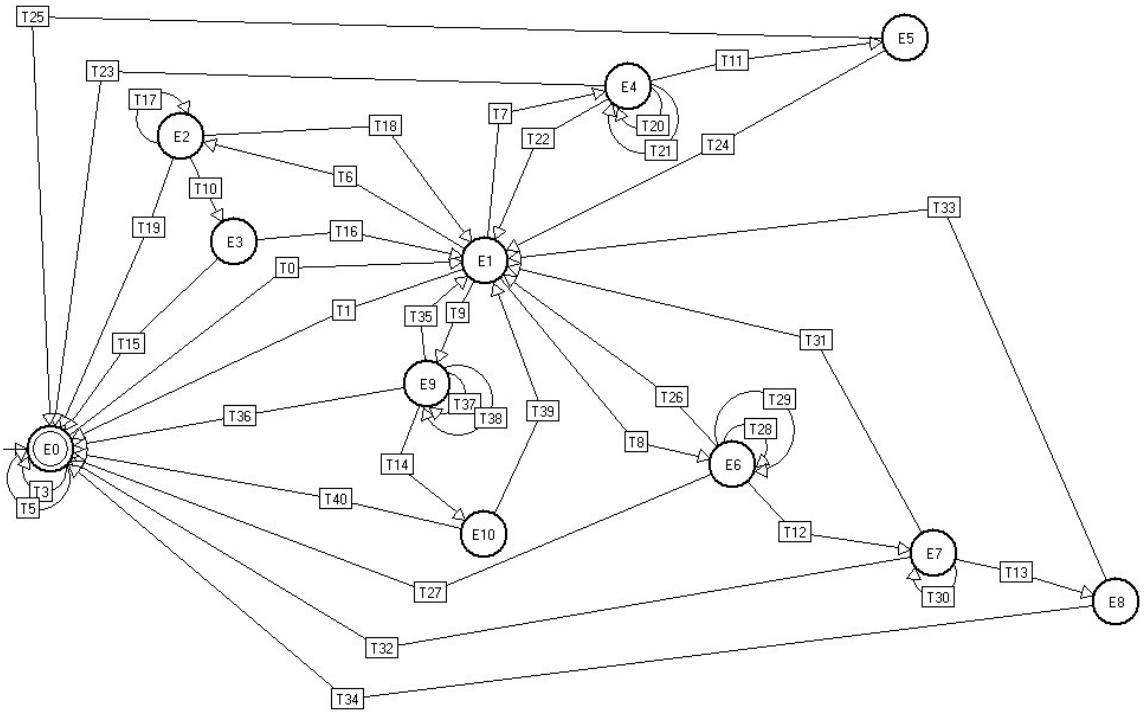

Figura 7.1: MFE que modela todos os cenários do IMEAD

Neste estudo de caso, a aplicação da STER resultou na redução significativa do número de casos de teste gerados, passando de mais de 23.000 para 9, como mostram os dados da Tabela 7.1. Essa redução significa que não foram gerados casos de teste para o sistema completo, entretanto, foram gerados casos de teste para os aspectos mais críticos do sistema. Isso pode ser útil, por exemplo, quando não houver tempo ou recursos disponíveis para testar todas as funcionalidades do sistema.

7.2 Estudo de caso 2: software de preparação de plano de vôo do SACI

O software de preparação de plano de vôo foi utilizado para verificar a aplicabilidade da STER a sistemas reais. Foram avaliados 3 aspectos:

1. **Quanto aos casos de uso e cenários apontados como críticos pela análise de riscos adotada pela STER:** Depois de ter sido feita a análise de riscos, os CdUs e cenários determinados como prioritários/críticos estavam de acordo com a expectativa da criadora.
2. **Casos de teste gerados pela STER X casos de testes gerados para as funções críticas do software:** Os 2 casos de teste gerados pela STER foram comparados com os 69 criados, de forma *ad-hoc*, para o software de preparação de plano de vôo do SACI. Nessa comparação percebeu-se que os casos de teste gerados pela STER **não** estavam entre os casos de teste criados para o software. O que significa que não haviam sido criados casos de teste para os casos de uso e cenários prioritários/críticos do software de preparação de plano de vôo do SACI.
3. **Análise dos 69 casos de testes criados, de forma *ad-hoc*, para o software de preparação de plano de vôo do SACI:** Dentre esses 69 casos de teste criados, 30 eram para testar funções relativas ao ator *Cientista* e 30 para o ator *Operador de Subsistema*. Como esses atores têm o mesmo perfil de acesso, poderia ter sido criado apenas um conjunto de casos de teste para testar os dois atores.

Como o objetivo não é criar casos de teste para o sistema completo e sim para as partes mais críticas do sistema, vimos que a STER foi de grande utilidade. Isto porque, além de restringir o número de casos de teste gerados, permitiu que eles fossem criados de forma sistemática e gerou testes para as partes mais críticas do sistema que tinham ficado de fora nos testes criados para o software de preparação de plano de vôo do SACI.

Capítulo 8

Conclusões

Nesse trabalho foi apresentada a STER, uma estratégia criada para auxiliar nos testes de sistemas reativos. Ela foi desenvolvida com o objetivo de atender à necessidade da indústria em contar com um método de geração de casos de teste efetivo que possa ser usado de forma sistemática, que gere casos de teste de forma automatizada usando ferramentas e que mostre que as partes críticas do sistema estão funcionando de acordo com o que foi especificado.

Com o intuito de encontrar informações que pudessem auxiliar na criação da STER foi realizado um estudo sobre testes funcionais baseados em casos de uso, modelos de estados e riscos. Além disso, foram feitas pesquisas sobre estratégias para geração de casos de teste que tratam desses temas.

Para atingir seus objetivos a STER é baseada na especificação formal do sistema modelada usando artefatos UML geralmente produzidos nas fases de análise e projeto. São usados casos de uso e diagramas de seqüência, que têm sido adotados por muitas indústrias de software, além das máquinas finitas de estados estendidas para permitir a geração automatizada dos casos de teste pela ConDado, uma ferramenta de geração de casos de teste.

Foi desenvolvida também uma análise de riscos que mostra quais são as partes mais críticas e vulneráveis do sistema, o que permite concentrar os esforços de teste para reduzir o risco de ocorrerem falhas nessas partes.

A STER teve ainda que contornar algumas limitações da ConDado para permitir a geração automatizada de testes. Essa ferramenta (i) gera um grande número de casos de teste, (ii) gera testes não executáveis e (iii) necessita que a máquina finita de estados estendida seja traduzida para uma linguagem própria para que os casos de teste possam ser gerados. Para mitigar o primeiro problema foi desenvolvida uma técnica de análise de riscos, o que ajudou a reduzir significativamente o número de casos de teste gerados. Para reduzir o número de testes não executáveis gerados foi usada a parametrização de dados.

Para auxiliar na tarefa de tradução da MFEE foi usada uma ferramenta de apoio, o Modelador de Máquina de Estados, que traduz a MFEE gráfica para a linguagem usada pela ConDado.

A fim de verificar sua eficácia, a STER foi usada em dois estudos de caso: (1) uma aplicação web que serviu como exemplo e (2) uma aplicação web real de pequeno porte da área espacial. Levando-se em consideração os dois exemplos que foram usados para avaliação da STER, chegou-se à conclusão que a STER é útil para a geração de casos de teste para as partes críticas do sistema e reduz significativamente o número de casos de testes gerados. *Entretanto, ela não deve ser usada como única fonte de testes, pois os casos de teste gerados são restritos a algumas partes do sistema. Como proposto, ela deve ser usada na fase de testes de sistema, após terem sido executados os testes de unidade, integração e validação*.

8.1 Contribuições da dissertação

Esse trabalho propôs uma estratégia de teste para auxiliar nos testes de sistemas reativos com o objetivo de mitigar alguns dos problemas que impedem que os casos de testes sejam gerados sistematicamente pela indústria e abrir caminho para que outras soluções possam ser encontradas. A seguir são mostrados os problemas encontrados e o que foi proposto para mitigar cada um deles.

- **Necessidade de uma especificação estável, o que é difícil de se obter na prática:** A STER divide o sistema em casos de uso. Portanto, os testes podem ser considerados de forma incremental, onde só os casos de uso completamente especificados são testados primeiro. À medida que o sistema vai sendo especificado as novas partes do sistema vão sendo testadas.
- **Uso de especificação informal que pode ser ambígua e dificultar o entendimento do sistema:** A STER torna a especificação mais precisa mostrando como transformar uma especificação descrita usando casos de uso em uma especificação formal usando máquina finita de estados.
- **Uso de métodos formais que são baseados em teorias matemáticas:** A STER usa artefatos UML para especificar formalmente os sistemas, uma vez que eles são amplamente usados na prática.
- **Explosão de estados que podem ocorrer devido aos parâmetros de interação de um sistema e devido à concorrência:** A STER concentra o esforço de teste nas partes do sistema onde as falhas podem causar um maior impacto no seu comportamento. Isso é feito usando testes baseados em riscos, onde só os casos de uso considerados mais críticos são detalhados de forma mais precisa e testados com mais atenção.

- **Geração de um grande número de casos de teste e criticidade de algumas partes do sistema que podem levar a danos significativos:** A STER define uma análise de riscos que ajuda determinar quais as partes do sistema são mais críticas e, portanto, quais devem ser testadas com mais atenção. Isso possibilita a redução do número de casos de teste gerados além de focar os testes nas partes mais críticas do sistema.
- **Número de entradas possíveis do sistema que pode ser intratável:** A STER usa parametrização dos dados para reduzir o número de entradas. Essa parametrização determina o número exato de vezes que cada entrada será executada.
- **Determinar quando os testes devem ser finalizados, o número de testes que devem ser aplicados e em qual ordem eles devem ser executados:** Esses são três dos grandes problemas encontrados durante os testes de sistemas. A STER usa a análise de riscos para selecionar as funcionalidades mais críticas do sistema, que devem ser testadas primeiro. Caso ainda existam tempo e recursos depois dos testes das funcionalidades críticas deve-se selecionar outras partes do sistema para serem testadas. Além disso, são criadas seqüências de casos de uso baseadas nas restrições e dependências em relação à sua execução.

  Assim, é possível determinar quando os testes serão finalizados (devem ser finalizados quando todas as partes críticas tiverem sido testadas, por exemplo); quantos testes devem ser aplicados (devem ser aplicados todos os testes relativos às partes críticas, por exemplo); e por fim, em qual ordem eles devem ser executados (devem ser executados primeiro os testes das partes críticas, começando pelo caso de uso que inicia a execução da seqüência, por exemplo).
- **Inexperiência dos testadores que tendem a executar os mesmos caminhos no software não verificando assim nenhuma funcionalidade adicional do sistema:** A STER gera os casos de teste percorrendo os caminhos de uma máquina finita de estados. Isso faz com que sejam gerados casos de teste que executem funcionalidades diferentes no software, independentemente da experiência do testador.

Resumindo, as principais contribuições desse trabalho são:

- Define uma estratégia de teste – a STER – que permite a geração automatizada de casos de testes usando artefatos produzidos na fase de análise e que pode ser usada sistematicamente pela indústria.
- Mostra, passo a passo, como os casos de teste podem ser gerados a partir da especificação (em casos de uso) do sistema.
- Define, passo a passo, uma técnica de análise de riscos para testes que leva em consideração a finalidade que o sistema tem para o usuário.

8.2 Trabalhos futuros

A STER ainda pode ser incrementada para gerar casos de testes de forma a verificar outros aspectos do sistema. Ela pode ser estendida para:

- Gerar testes para seqüências de casos de uso intercaladas, ou seja, gerar apenas uma seqüência de CdUs com as várias instâncias para o mesmo parâmetro evitando assim, a duplicação dos casos de uso instanciados.
- Gerar testes combinando o uso do sistema por vários atores simultaneamente verificando assim como o sistema se comporta com acessos concorrentes.
- Usar técnicas de análise de risco que permita analisar a segurança (*safety*) dos sistemas.
- Gerar um número mínimo de seqüências eliminando casos de teste que não cubram novas funcionalidades.

Deve ser verificado ainda:

- Como a STER se comporta em relação a sistemas não reativos
- Como a STER se comporta em relação a sistemas reativos que não sejam aplicações *web*
- Como a STER se comporta em relação a sistemas de médio e grande porte, uma vez que foram usados como estudos de caso apenas sistemas de pequeno porte.
- Qual é o risco residual após o uso da STER e quais testes devem ser feitos caso o risco residual seja alto.

Referências

[Amland, 2000] Amland, S., (2000), "Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study", The Journal of Systems and Software, vol. 53, pp. 287-295

[Ardis+, 1996] Ardis, M. A., Chaves, J. A., Jagadeesan, L. J., et al, (1996), "A Framework for Evaluating Specification Methods for Reactive Systems", IEEE Transactions on Software Engineering.

[Bertolino+, 2004a] Bertolino, A., Marchetti, E., Muccini, H., (2004), "Introducing a Reasonably Complete and Coherent Approach for Model-Based Testing", Intlrnational Workshop on Test and Analysis of Component Based Systems (Tacos), Barcelona, Espanha.

[Bertolino+, 2004b] Bertolino, A., Gnesi, S., (2004), "PLUTO: A test Methodology for Product Families", LNCS 3014, pp. 181-197.

[Binder, 2000] Binder, R. V., (2000), "Testing Object-Oriented Systems: Models, Patterns, and Tools", Addison-Wesley.

[Booch+, 2000] Booch, G., Rumbaungh, J., Jacobson, I., (2000), "UML: Guia do usuário", Ed. Campus.

[Briand+, 2002] Briand, L., Labiche, Y., (2002), "A UML-based Approach to System Testing", Carleton University.

[Bousquet+, 1999] Bousquet, L., Ouabdesselam, F., Richier, J.-L., Zuanon, N., (1999), "Lutess: a Specification-Driven Testing Environment for Synchronous Software".

[Chen, 2002] Chen, Y., (2002), "Specification-Based Regression Testing Measurement with Risk Analysis", Dissertação de mestrado. University of Ottawa, Canadá.

[Dilonardo, 1990] Dilonardo, F., (1990), "A comparison between conformance, interoperability, performance, development testing, conformance testing and certification". Information Technology and Telecommunications, IOS Press.

[Farias, 2003] Farias, C. M., (2003), "Um Método de Teste Funcional para Verificação de Componentes", Dissertação de mestrado, UFCG, Campina Grande, Paraíba, Brasil.

[Grabowski, 1994] Grabowski, J., (1994), "SDL and MSC Based Test Case Generation – An Overall View of the SAMSTAG Method", Relatório técnico IAM-94-005, Universität Bern, http://www.iam.unibe.ch/publikationen/techreports/1994/iam-94-005

[Halbwachs, 1998] Halbwachs, N., (1998), "Synchronous Programming of Reactive Systems: A Tutorial and Commented Bibliography", 10º Conferência Internacional em Computer-Aided Verification, CAV'98, Vancouver, LNCS 1427, Springer Verlag, http://citeseer.ist.psu.edu/10686.html.

[Harel+, 1998] Harel, D., Politi, M., (1998), "Modeling Reactive Systems with Statecharts – The Statemate Approach", McGraw-Hill.

[Hyatt+, 1996] Hyatt, L., Rosenberg, L., (1996), "Software Quality Model and Metrics for Risk Assessement", European Space Agency Software Assurance Symposium, Netherlands e 8º Annual Software Technology Conference, Utah, http://satc.gsfc.nasa.gov/support/ESA_MAR96/quality/esa_qual.html

[Jacobson+, 1992] Jacobson, I., Christerson, M., Jonsson, P., Öergaard, G., (1992), "Object-Oriented Software Engineering: a Use Case Driven Approach", Adisson-Wesley.

[Jagadeesan+, 1995] Jagadeesan, L.J., Puchol, C., Olnhausen, J. E. V., (1995), "Safety Property Verification of ESTEREL Programs and Applications to Telecommunications Software". Proc. of the 7th International Conference on Computer-Aided Verification.

[Jagadeesan+, 1997a] Jagadeesan, L.J., Porter, A., Puchol, C., Ramming, J. C., Votta, L. G., (1997), "Specification-based Testing of Reactive Software: A Case Study in Technology Transfer". Proc. of the 19th International Conference on SE.

[Jagadeesan+, 1997b] Jagadeesan, L.J., Porter, A., Puchol, C., Ramming, J. C., Votta, L. G., (1997), "Specification-based Testing of Reactive Software: Tools and Experiments", Experience Report, Proc. of the 19th International Conference on SE.

[Lai+, 1995] Lai, R., Leung, W. (1995), "Industrial and academic protocol testing: the gap and the means of convergence". Computer Networks and ISDN Systems, 27, pp537-547.

[Martins+, 2004] Martins, E., Guimarães, D. C., Ambrosio, A. M., (2004), "STER: A Strategy for Testing Reactive Systems", Conferência Internacional em Dependable Systems and Network – DSN.

[Martins, 2003] Martins, E. (2003), "Qualidade de Software – V&V – Introdução", Notas de aula.

[Martins, 2003b] Martins, E. (2003), "O projeto ATIFS: uma visão geral", V Workshop do Projeto ATIFS – Ambiente de Teste baseado em Injeção de Falhas por Software, Unicamp.

[Martins+, 1999] Martins, E., Sabião, S.B., Ambrosio, A. M., (1999), "ConData: A Tool for Automating Specification-based Test Case Generation for Communication Systems", Software Quality Journal, Vol. 8, No.4, 303-319, edited by Anna Liu and Paddy Nixon - Kluwer Academic Publishers.

[Martins, 1995] Martins, E., (1995), "ATIFS: Um Ambiente de Testes baseado em Injeção de Falhas por Software", Relatório Técnico, DCC-95-24.

[McGregor+, 2000] McGregor, J.D., Major, M.L., (2000), "Selecting Test Cases Based on User Priorities".

[Melo, 2003] Melo, P. C. B., Junior, R. A. P. (2003), "Ferramenta de Especificação Gráfica de Máquinas de Estados Finitas para o Ambiente de Teste baseado em Injeção de Falhas por Software (ATIFS)", Relatório Final, UFRN, Natal, Rio Grande do Norte, Brasil.

[Mosley, 2000] Mosley, D. J., (2000), "Client-Server Software Testing on the Desktop and the Web", Prentice Hall.

[Murray+, 1997] Murray, L., Carrington, D., MacColl, I. e Strooper, P. (1997), "Extending Test Templates with Inheritance".

http://citeseer.ist.psu.edu/murray97extending.html.

[Pereira Junior+, 2000] Pereira Junior, R. A., Francisco, M. F. M. e Carvalho, M. J. M. (2000), "Building a telecommand web system for scientific microsatellite control based on open source technology", 30th International Symposium – SPACE OPS 2000, France.

[Perry, 1995] Perry, W. (1995), "Effective methods for software testing", John Wiley & Sons, Inc.

[Pressman, 1997] Pressman, R. S. (1997), "Software Engineering – A practitioner's Approach". McGraw-Hill, 5ª ed.

[Regnell+, 2000] Regnell, B., Runeson, P., Wohlin, C., (2000), "Towards Integration of Use Case Modelling and Usage-Based Testing", Journal of Systems and Software, Vol. 50, pp.117-130.

http://www.tts.lth.se/Personal/bjornr/Papers/JSS00.pdf

[Rosenberg+, 1996] Rosenberg, L. H., Hyatt, L. E., (1996), "Software Metrics Program for Risk Assessment", 47º International Astronautical Congress & Exhibition, 29º Safety and Rescue Symposium, China.

http://satc.gsfc.nasa.gov/support/IAC_OCT96/iaf.html

[Rumbaugh+, 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen W., (1991), "Object-Oriented Modelling and Design", Prentice-Hall International.

[Ryser+, 2000] Ryser, J. Glinz, M. (2000), "SCENT: A Method Employing Scenarios to Systematically Derive Test Cases for System Test", Technical Report, Institut für Informatik, Universität Zürich.

[Sabiao, 1998] Sabião, S. (1998), "Um método para Geração de Testes baseado em Máquina Finita de Estado Estendida Combinando Técnicas de Teste Caixa Preta", Dissertação de Mestrado, Instituto de Computação, UNICAMP, Campinas, São Paulo, Brasil.

[Sommerville, 2001] Sommerville, I., (2001), "Software Engineering", Addison-Wesley, 6ª ed.

[Ural, 1992] Ural, H., (1992), "Formal methods for test sequence generation", Computer Communications, vol. 15, nº 5, pp. 311-325.

[Volpi, 2001] Volpi, L.M., (2001), “Uma Estratégia de Teste de Software para Ambiente Cliente-Servidor”, Dissertação de mestrado, UFPR, Curitiba, Paraná, Brasil.

[Warmer+, 1999] Warmer, J., Kleppe, A., (1999), “The Object Constraint Language”, Addison-Wesley.

[Wittevrongel+, 2001a] Wittevrongel, J., Maurer, F., (2001), “SCENTOR: Scenario-Based Testing of E-Business Applications”, 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2001), USA, IEEE Computer Society, pp. 41-48.

http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/wetice/&toc=comp/proceedings/wetice/2001/1269/00/1269toc.xml&DOI=10.1109/ENABL.2001.953386

[Wittevrongel+, 2001b] Wittevrongel, J., Maurer, F., (2001), “Using UML to Partially Automate Generation of Scenario-Based Test Drivers”, 7th International Conference on Object Oriented Information Systems (OOIS'2001), Canada, Springer, pp. 303-306.

Apêndice A

Aplicação da STER ao IMEAD

Neste apêndice são mostrados os detalhes da aplicação da STER ao estudo de caso IMEAD, apresentado no capítulo 5.

Nas seções A.1 e A.2 são descritos, respectivamente, os atores e os casos de uso do IMEAD. A seção A.3 mostra o cálculo da exposição ao risco dos casos de uso. Na seção A.4 são mostradas as seqüências de casos de uso parametrizadas. A seção A.5 apresenta o cálculo da exposição ao risco dos cenários e por fim a seção A.6 mostra os casos de testes gerados para o IMEAD.

A.1 Descrição dos atores

O IMEAD possui 3 atores: administrador, professor e aluno. Eles estão descritos nas tabelas A.1, A.2 e A.3, respectivamente.

Tabela A.10: Descrição do ator *administrador*

Ator: Administrador **Descrição:** Gerencia o sistema IMEAD **Nível de habilidade:** Experiente **Perfil do ator:**	
Caso de uso	**Freqüência relativa de uso**
1– Entrar no sistema	Alta
2– Cadastrar usuário	Alta
3– Excluir usuário	Baixa
4– Alterar dados	Baixa
5– Visualizar Dados	Média
6– Sair do sistema	Alta
7– Entrar no curso	N/A

Tabela A.11: Descrição do ator *professor*

Ator: Professor **Descrição:** Ministra o curso cadastrado no sistema **Nível de habilidade:** Treinado, mas com experiência variada. **Perfil do ator:**	
Caso de uso	**Freqüência relativa de uso**
1– Entrar no sistema	Alta
2– Cadastrar usuário	N/A
3– Excluir usuário	N/A
4– Alterar dados	Baixa
5– Visualizar Dados	Média
6– Sair do sistema	Alta
7– Entrar no curso	Alta

Tabela A.12: Descrição do ator *aluno*

Ator: Aluno **Descrição:** Faz o curso cadastrado no sistema **Nível de habilidade:** Variado **Perfil do ator:**	
Caso de uso	**Freqüência relativa de uso**
1– Entrar no sistema	Alta
2– Cadastrar usuário	N/A
3– Excluir usuário	N/A
4– Alterar dados	Baixa
5– Visualizar Dados	Média
6– Sair do sistema	Alta
7– Entrar no curso	Alta

A.2 Descrição dos casos de uso

O IMEAD possui 7 casos de uso (CdUs). Entretanto, nem todos eles foram considerados prioritários. Devido a restrições de espaço, só serão detalhados os casos de uso/cenários para os quais foram gerados casos de teste. Os demais são apenas apresentados. Além disso, são apresentados apenas os casos de uso do ator *administrador*, que será usado como exemplo.

As tabelas A.5 e A.7 mostram os casos de uso prioritários com todos os seus cenários. As tabelas A.4 e A.9 mostram os casos de uso que estão na seqüência de casos de uso. Nessas tabelas são detalhados apenas os cenários que ativam a execução do cenário seguinte na seqüência ou são ativados por um cenário da seqüência. As tabelas A.6 e A.8 mostram os casos de uso que não estão na seqüência de CdUs, por isso não foi detalhado nenhum cenário.

Tabela A.13: Caso de uso *Entrar no Sistema*

Caso de uso 1 **Nome:** Entrar no sistema **Descrição:** Permite que o usuário entre no sistema. **Autores:** Daniele Constant Guimarães **Atores:** Administrador, professor, aluno. **Prioridade:** A ser calculada **Pré-condição:** O usuário está na página inicial do sistema. **Pós-condição:** A página com as opções do usuário é exibida corretamente. **Parâmetros:** entrada: senhaUsr: string; entrada/saída: matriculaUsr: string; saída: categoriaUsr: string
Ator: administrador
Fluxo normal: **Cenário 1:** 1. O usuário digita a matrícula e senha corretas. 2. O sistema exibe a página que contém as opções do usuário.

Tabela A.14: Caso de uso *Cadastrar Usuário*

Caso de uso 2 **Nome:** Cadastrar usuário **Descrição:** Cadastro de novos usuários do sistema. **Autores:** Daniele Constant Guimarães **Atores:** Administrador **Prioridade:** A ser calculada **Pré-condição:** O usuário está na página de opções. **Pós-condição:** Os dados do novo usuário são cadastrados com sucesso **Parâmetros:** entrada/saída: matriculaUsr, categoriaUsr: string; entrada: matriculaCadastro: string
Ator: administrador
Fluxo normal: **Cenário 2:** 1. O usuário escolhe a opção cadastrar usuários. 2. O sistema exibe a página para cadastrar usuários. 3. O usuário digita os dados completos. 4. O sistema confirma o cadastro. 5. O usuário escolhe a opção voltar para a página de opções. 6. O sistema exibe a página que contém as opções do usuário. **Fluxo alternativo:** **Cenário 3:** 3a. O usuário cancela a operação de cadastro. 3a.1. Volta à página de opções. 3a.2. Sai do caso de uso. **Cenário 4:** 3b. O usuário sai do sistema. 3b.1. Vá para o caso de uso 6. **Cenário 5:** 5a. O usuário sai do sistema. 5a.1. Vá para o caso de uso 6. **Fluxo de exceção:** **Cenário 6:** 3c. O usuário não informou todos os dados necessários. 3c.1. O sistema exibe uma mensagem informando que os dados estão incompletos. 3c.2. Retorne ao passo 3

Tabela A.15: Caso de uso *Alterar Dados*

Caso de uso 4 **Nome:** Alterar dados **Descrição:** Altera dados do usuário. **Autores:** Daniele Constant Guimarães **Atores:** Administrador, professor, aluno. **Prioridade:** A ser calculada **Pré-condição:** O usuário está na página de opções. **Pós-condição:** Os dados do usuário são alterados com sucesso. **Parâmetros:** entrada/saída: matriculaUsr, categoriaUsr: string; entrada: matriculaAlterar: string

Tabela A.16: Caso de uso *Excluir Usuário*

Caso de uso 3 **Nome:** Excluir usuário **Descrição:** Exclui usuários do sistema. **Autores:** Daniele Constant Guimarães **Atores:** Administrador **Prioridade:** A ser calculada **Pré-condição:** O usuário está na página de opções. **Pós-condição:** Os dados do usuário são excluídos com sucesso. **Parâmetros:** entrada/saída: matriculaUsr, categoriaUsr: string; entrada: matriculaExclusao: string
Ator: administrador
Fluxo normal: **Cenário 7:** 1. O usuário escolhe a opção excluir usuário. 2. O sistema exibe a página para exclusão de usuários. 3. O usuário digita a matrícula para excluir o usuário. 4. O sistema confirma a exclusão. 5. O usuário escolhe a opção voltar para a página de opções. 6. O sistema exibe a página que contém as opções do usuário. **Fluxo alternativo:** **Cenário 8:** 3a. O usuário cancela a operação de exclusão. 3a.1. Volta à página de opções. 3a.2. Sai do caso de uso. **Cenário 9:** 3b. O usuário sai do sistema. 3b.1. Vá para o caso de uso 6. **Cenário 10:** 5a. O usuário sai do sistema. 5a.1. Vá para o caso de uso 6. **Fluxo de exceção:** **Cenário 11:** 3c. O usuário digitou uma matrícula inexistente. 3c.1. O sistema exibe uma mensagem informando que a matrícula não existe. 3c.2. Retorne ao passo 3. **Cenário 12:** 3d. O usuário digitou a matrícula incorreta. 3d.1. Retorne ao passo 3.

Tabela A.17: Caso de uso *Visualizar Dados*

Caso de uso 5 **Nome:** Visualizar dados **Descrição:** Permite visualizar os dados dos usuários. **Autores:** Daniele Constant Guimarães **Atores:** Administrador, professor, aluno. **Prioridade:** A ser calculada **Pré-condição:** O usuário está na página de opções. **Pós-condição:** Os dados do usuário são exibidos com sucesso. **Parâmetros:** entrada/saída: matriculaUsr, categoriaUsr: string; entrada: matriculaVisualizar: string

Tabela A.18: Caso de uso *Sair do Sistema*

Caso de uso 6 **Nome:** Sair do sistema **Descrição:** O usuário sai do sistema. **Autores:** Daniele Constant Guimarães **Atores:** Administrador, professor, aluno. **Prioridade:** A ser calculada **Pré-condição:** O usuário está em qualquer página do sistema. **Pós-condição:** O usuário sai do sistema corretamente. **Parâmetros:** entrada: matriculaUsr, categoriaUsr: string
Ator: administrador
Fluxo normal: **Cenário 13:** 1. O usuário escolhe a opção sair do sistema. 2. O sistema exibe a página inicial do sistema.

A.3 Cálculo da exposição ao risco dos casos de uso

Para calcular a exposição ao risco dos casos de uso deve-se usar (i) a lista de indicadores de riscos mostrada na Tabela 5.12, no capítulo 5 e (ii) a fórmula $\mathrm{Re}(f) = \sum_{j=1}^{n} i_j * w_j$, apresentada na seção 5.2. A seguir são mostrados os cálculos para cada um dos casos de uso do IMEAD. O resultado desses cálculos é mostrado na Tabela 5.13, no capítulo 5.

$$\mathrm{Re}(EntrarNoSistema) = \sum_{j=1}^{7} i_j * w_j = (1\times 2)+(3\times 1)+(1\times 3)+(1\times 2)+(2\times 5)+(3\times 2)+(1\times 1) = 27$$

$$\mathrm{Re}(CadastrarUsuario) = \sum_{j=1}^{7} i_j * w_j = (2\times 2)+(2\times 1)+(2\times 3)+(3\times 2)+(3\times 5)+(3\times 2)+(2\times 1) = 41$$

$$\mathrm{Re}(ExcluirUsuario) = \sum_{j=1}^{7} i_j * w_j = (2\times 2)+(2\times 1)+(2\times 3)+(2\times 2)+(3\times 5)+(1\times 2)+(2\times 1) = 35$$

$$\mathrm{Re}(AlterarDados) = \sum_{j=1}^{7} i_j * w_j = (3\times 2)+(2\times 1)+(2\times 3)+(2\times 2)+(2\times 5)+(1\times 2)=(2\times 1) = 32$$

$$\mathrm{Re}(VisualizarDados) = \sum_{j=1}^{7} i_j * w_j = (3\times 2)+(2\times 1)+(2\times 3)+(1\times 2)+(1\times 5)+(2\times 2)+(1\times 1) = 26$$

$$\mathrm{Re}(SairDoSistema) = \sum_{j=1}^{7} i_j * w_j = (1\times 2)+(0\times 1)+(1\times 3)+(1\times 2)+(1\times 5)+(3\times 2)+(1\times 1) = 19$$

A.4 Definição das seqüências de casos de uso parametrizadas

As seqüências de casos de uso parametrizadas do IMEAD são definidas a partir do diagrama de atividades da Figura 5.4 mostrada na seção 5.4.

Primeiro, foi criado um grafo direcionado que corresponde ao diagrama de atividades da Figura 5.4. Esse grafo é mostrado na Figura A.1. Nele, o nó *A* corresponde à atividade *A* (*Entrar no sistema*) do diagrama de atividades, o nó *B* responde à atividade *B* (*Cadastrar usuário*) e assim por diante. Esse grafo foi obtido transformando os "*joins*" e "*forks*" em arestas regulares, como por exemplo o *fork* da atividade *B* para as atividades *C*, *D* e *E* na Figura 5.4 foi transformado nas arestas dos nós *B* para *C*, *B* para *D* e *B* para *E* na Figura A.1.

Para encontrar as seqüências de casos de uso a partir desse grafo direcionado foi feita uma busca em profundidade que levou em consideração a restrição de execução dos *loops* (eles devem ser executados no máximo uma vez). Os caminhos encontrados pela busca em profundidade representam as seqüências de CdUs. Um dos caminhos encontrados foi *Início.A.B.B.C.C.F.Fim*, que correspondente à seqüência de CdUs:

- EntrarNoSistema.CadastrarUsuario.ExcluirDados.SairDoSistema
  (Seqüência 1)

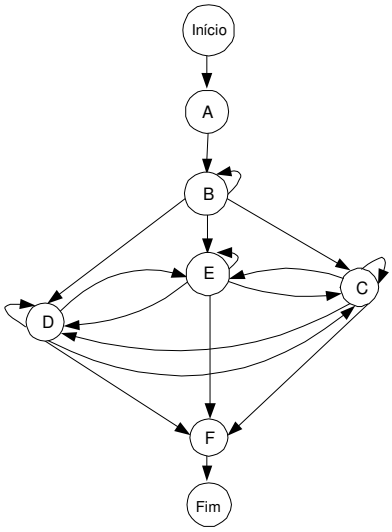

Figura A.2: Grafo direcionado

A.5 Cálculo da exposição ao risco dos cenários

Para calcular a exposição ao risco dos cenários, assim como foi feito para os casos de uso, deve-se usar (i) a lista de indicadores de riscos mostrada na Tabela 5.12, no capítulo 5 e (ii) a fórmula $\mathrm{Re}(f) = \sum_{j=1}^{n} i_j * w_j$, apresentada na seção 5.2. A seguir são mostrados os cálculos para cada um dos cenários que pertencem aos casos de uso prioritários do IMEAD. O resultado desse cálculo é mostrado na Tabela 5.14, no capítulo 5.

$$\mathrm{Re}(cenário2) = \sum_{j=1}^{7} i_j * w_j = (2\times2)+(2\times1)+(3\times3)+(3\times2)+(3\times5)+(3\times2)+(2\times1) = 44$$

$$\mathrm{Re}(cenário3) = \sum_{j=1}^{7} i_j * w_j = (3\times2)+(3\times1)+(1\times3)+(1\times2)+(3\times5)+(1\times2)+(2\times1) = 33$$

$$\mathrm{Re}(cenário4) = \sum_{j=1}^{7} i_j * w_j = (3\times2)+(0\times1)+(1\times3)+(1\times2)+(1\times5)+(1\times2)+(2\times1) = 20$$

$$\mathrm{Re}(cenário5) = \sum_{j=1}^{7} i_j * w_j = (3\times2)+(0\times1)+(1\times3)+(1\times2)+(1\times5)+(1\times2)+(2\times1) = 20$$

$$\mathrm{Re}(cenário6) = \sum_{j=1}^{7} i_j * w_j = (2\times2)+(2\times1)+(3\times3)+(3\times2)+(3\times5)+(2\times2)+(2\times1) = 42$$

$$\mathrm{Re}(cenário7) = \sum_{j=1}^{7} i_j * w_j = (3\times2)+(2\times1)+(3\times3)+(3\times2)+(3\times5)+(3\times2)+(2\times1) = 46$$

$$\mathrm{Re}(cenário8) = \sum_{j=1}^{7} i_j * w_j = (3\times2)+(3\times1)+(1\times3)+(1\times2)+(3\times5)+(1\times2)+(2\times1) = 33$$

$$\mathrm{Re}(cenário9) = \sum_{j=1}^{7} i_j * w_j = (3\times2)+(0\times1)+(1\times3)+(1\times2)+(1\times5)+(1\times2)+(2\times1) = 20$$

$$\mathrm{Re}(cenário10) = \sum_{j=1}^{7} i_j * w_j = (3\times2)+(0\times1)+(1\times3)+(1\times2)+(1\times5)+(1\times2)+(2\times1) = 20$$

$$\mathrm{Re}(cenário11) = \sum_{j=1}^{7} i_j * w_j = (3\times2)+(3\times1)+(3\times3)+(2\times2)+(2\times5)+(2\times2)+(2\times1) = 38$$

$$\mathrm{Re}(cenário12) = \sum_{j=1}^{7} i_j * w_j = (3\times2)+(3\times1)+(3\times3)+(2\times2)+(3\times5)+(2\times2)+(2\times1) = 43$$

A.6 Casos de teste gerados

Usando as ferramentas ConDado (Controle e Dados) e MME (Modelador de Máquina de Estados), ambas descritas no capítulo 4, foram gerados 9 casos de teste para o IMEAD. Esses casos de teste foram transcritos a seguir.

Caso de teste 1:
DigitaMatSenhaCorretas (matu1,senu1,catu1)
EscolheOpcaoSair (matu1,catu1)

Caso de teste 2:
DigitaMatSenhaCorretas (matu1,senu1,catu1)
EscolheOpcaoCadastrar (matu1,catu1)
DigitaDadosCompletos (matu1,catu1,matCadCor1)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoSair (matu1,catu1)

Caso de teste 3:
DigitaMatSenhaCorretas (matu1,senu1,catu1)
EscolheOpcaoCadastrar (matu1,catu1)
DigitaDadosCompletos (matu1,catu1,matCadCor1)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoExcluir (matu1,catu1)
DigitaMatriculaCorreta(matu1,catu1,matExcCor)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoSair (matu1,catu1)

Caso de teste 4:
DigitaMatSenhaCorretas (matu1,senu1,catu1)
EscolheOpcaoCadastrar (matu1,catu1)
DigitaDadosCompletos (matu1,catu1,matCadCor1)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoExcluir (matu1,catu1)
DigitaMatriculaIncorreta (matu1,catu1,matExcInc)
DigitaMatriculaCorreta (matu1,catu1,matExcCor)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoSair (matu1,catu1)

Caso de teste 5:
DigitaMatSenhaCorretas (matu1,senu1,catu1)
EscolheOpcaoCadastrar (matu1,catu1)
DigitaDadosIncompletos (matu1,catu1,matCadInc1) /ExibeMensagem
DigitaDadosCompletos (matu1,catu1,matCadCor1)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoSair (matu1,catu1)

Caso de teste 6:
DigitaMatSenhaCorretas (matu1,senu1,catu1)
EscolheOpcaoExcluir (matu1,catu1)
DigitaMatriculaCorreta (matu1,catu1,matExcCor)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoSair (matu1,catu1)

Caso de teste 7:
DigitaMatSenhaCorretas (matu1,senu1,catu1)
EscolheOpcaoExcluir (matu1,catu1)
DigitaMatriculaCorreta (matu1,catu1,matExcCor)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoCadastrar (matu1,catu1)
DigitaDadosCompletos (matu1,catu1,matCadCor1)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoSair (matu1,catu1)

Caso de teste 8:
DigitaMatSenhaCorretas (matu1,senu1,catu1)
EscolheOpcaoExcluir (matu1,catu1)
DigitaMatriculaCorreta (matu1,catu1,matExcCor)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoCadastrar (matu1,catu1)
DigitaDadosIncompletos (matu1,catu1,matCadInc1) /ExibeMensagem
DigitaDadosCompletos (matu1,catu1,matCadCor1)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoSair (matu1,catu1)

Caso de teste 9:
DigitaMatSenhaCorretas (matu1,senu1,catu1)
EscolheOpcaoExcluir (matu1,catu1)
DigitaMatriculaIncorreta (matu1,catu1,matExcInc)
DigitaMatriculaCorreta (matu1,catu1,matExcCor)
EscolheOpcaoVoltarPgOpcoes (matu1,catu1)
EscolheOpcaoSair (matu1,catu1)

Anexo A

Depoimento

"A STER é útil para o que se propõe. Ou seja, ela gera casos de teste para verificar se as partes críticas do sistema estão funcionando corretamente. Entretanto, não deve ser usada como única fonte de testes, pois não gera casos de teste para o sistema como um todo."

Maria de Fátima Matiello Francisco
Criadora do software de preparação do plano de vôo do SACI