Patterns to Teach Software Testing to Non-developers

JULIANA HERBERT, Universidade Federal de Ciências da Saúde de Porto Alegre

This paper presents four patterns to teach software testing to non-developers. These patterns were derived from our experiences, as professor and as instructor, on teaching software testing techniques and strategies to professionals without software development background. We´ve proposed these patterns considering the context driven approach to software testing, using the risk-based strategy and exploratory testing combined with techniques using the black box testing approach. This approach is focused on the interaction of the software with the world, without considering the implementation. The value of the software functionalities behavior is the most important principle. To define the patterns, we were inspired by the "pedagogical patterns proposals", which try to capture expert knowledge of the practice of teaching and learning. Although we´ve developed these experiences on the context of the health domain software, we suppose we will be able to use those patterns on other critical or non-critical domains as well. These patterns can help professors and instructors to teach software testing in a meaningful way, changing behavior and not just learning concepts.

Categories and Subject Descriptors: **D.2.4 [Software/Program Verification]**: Validation; **J.3 [Life and Medical Sciences]**: Medical information systems; **I.5.2 [Design Methodology]**: Pattern analysis

General Terms: Human Factors

Additional Key Words and Phrases: User acceptance testing, context-driven testing, black box testing, pedagogical patterns.

1. INTRODUÇÃO

Há vários fatores que contribuem para o sucesso de empresas de desenvolvimento de software. Por exemplo, a consideração da qualidade nos processos de desenvolvimento, a preocupação com a qualidade do produto de software final e com os produtos intermediários ou complementares do projeto, a excelência na prestação de serviços associados ao software e a implantação de programas de capacitação de recursos humanos que compõem a organização.

Em geral este sucesso é obtido quando o conjunto de fatores mencionados é considerado de forma integrada, conciliando e balanceando a complementação e o conflito de forças relacionadas a estas iniciativas. Entretanto, caso o software não funcione, dificilmente os outros critérios serão considerados.

Uma das formas mais utilizadas para validar a funcionalidade do software é o teste de software. Além do objetivo "clássico" do teste de encontrar defeitos (Myers 1979), outro objetivo muito importante é a investigação técnica e empírica do software em teste, conduzida para fornecer informações relevantes aos *stakeholders* (Kaner 2009).

Além das características da complexidade inerente às atividades de teste de software, deve-se também considerar que é impossível assegurar que um programa não contém falhas, o que faz com que seja também muito difícil determinar o fim das atividades de teste de software. A ideia de teste exaustivo é inviável (ISTQB 2011). Por isto, várias estratégias e técnicas de teste têm sido propostas na literatura com o objetivo de selecionar casos de teste confiáveis (casos que revelam defeitos). O objetivo final destes estudos é de encontrar o maior número de defeitos possível com o menor conjunto possível de casos de teste. Assim pode-se minimizar o esforço de teste.

Entre as várias categorias ou tipos de técnicas de teste propostas na literatura, as mais conhecidas (Marick 1995) são as do teste funcional, também chamado de teste de caixa-preta (*black box*) e as do teste estrutural, ou teste de caixa aberta (*white box*). As técnicas de teste estrutural requerem conhecimento da implementação do software em teste, de sua estrutura interna e da linguagem de programação utilizada. Já as técnicas de teste funcional têm o foco no comportamento do software, na execução de suas funções, considerando as entradas submetidas e as saídas geradas. O teste funcional requer conhecimento de domínio, principalmente.

Apesar do teste funcional poder ser realizado tanto por usuários do sistema como por desenvolvedores ou testadores de software, no caso de domínios mais complexos e/ou críticos, como é o caso de sistemas de software desenvolvidos para a área da saúde, o conhecimento profundo do domínio e do contexto no qual o sistema será utilizado é fundamental para a validação do software. Por outro lado, muitas vezes as pessoas que têm este conhecimento não conhecem as técnicas e estratégias de teste funcional, o que faz com que dependam da sorte e de outros fatores subjetivos para gerar valor com o teste.

Os padrões descritos neste trabalho foram desenvolvidos no contexto das experiências de ensino de teste de software na Universidade Federal de Ciências da Saúde de Porto Alegre (UFCSPAa 2016). Na UFCSPA, há 15 cursos de graduação na área da saúde. Entre eles, o curso de Informática Biomédica, no qual há disciplinas da área de Engenharia de Software. Ainda que as atividades de desenvolvimento de software propriamente ditas estejam concentradas neste curso, definiu-se como prioritário o envolvimento de profissionais especialistas de outras áreas para as atividades de validação e verificação dos sistemas de software desenvolvidos.

Para que as atividades de validação e verificação fossem realizadas de forma sistemática, definiu-se um curso de teste funcional para ser ministrado para estes profissionais. E, a partir das definições e experiências com a realização do curso de teste para estes potenciais usuários finais, foram derivados alguns padrões que descrevem práticas mais comuns neste contexto, os quais podem auxiliar professores e instrutores no ensino do teste de software para profissionais sem conhecimento em desenvolvimento de software. A elaboração destes padrões foi feita utilizando como base os *pedagogical patterns* (Bergin 2012) (Laurillard 2012), os *testing patterns* (Marick 2016) e vários artigos apresentados nas conferências *Pattern Language of Programs* – PLOP (The Hillside Group 2016).

Sintetizando, uma vez que se identificou a importância do envolvimento de profissionais que tenham conhecimento do domínio e do contexto no qual o sistema será utilizado nas atividades de teste de software, identificou-se também a necessidade de treinamento destes profissionais em técnicas de teste, para que esta atividade fosse realizada de forma sistemática. A partir das experiências já citadas, foram definidos os padrões descritos neste trabalho, como forma de auxílio para professores e instrutores que ensinam técnicas de teste de software para profissionais que não são da área de desenvolvimento de software. Este aprendizado deve ser significativo, gerando mudança de comportamento e, como consequência, a realização de teste com maior valor agregado.

Este artigo apresenta quatro destes padrões definidos neste contexto. Foi organizado nas seguintes seções:

- a Seção 2 apresenta conceitos relacionados à categoria de técnicas de teste caixa-preta;
- na Seção 3, são apresentadas algumas razões para justificar o ensino de técnicas de teste funcional a profissionais que não desenvolvem software;
- já na Seção 4, são apresentados os quatro padrões para o ensino de teste de software para profissionais sem conhecimento da área de desenvolvimento de software;
- na Seção 5, são apresentadas as considerações finais deste trabalho;
- posteriormente, são listadas as principais referências bibliográficas utilizadas como base para o conteúdo apresentado neste artigo.

2. TÉCNICAS DE TESTE CAIXA-PRETA

As técnicas de teste caixa-preta são também conhecidas como técnicas de teste funcional, já que o seu foco é na funcionalidade do sistema de software sendo testado.

As funções são “o que” o sistema faz. Podem estar documentadas em especificações de requisitos, casos de uso, especificações formais, histórias de usuário, ou até mesmo não estar documentadas. Apenas implementadas, de acordo com o que os desenvolvedores entenderam que seriam as funções deste sistema.

No teste de caixa-preta, não é realizada a análise do código do programa ou de sua estrutura. Um conjunto de casos de teste, composto por entradas combinadas ou não, ou por situações de uso, é submetido ao software em teste para que seu comportamento possa ser analisado, sendo considerado adequado ou não de acordo com as saídas ou comportamento esperados ou especificados. Neste contexto, analisa-se a interação do software com o “mundo”. O testador define os casos de teste com base em seu conhecimento das características do produto, das necessidades dos usuários, da área na qual é ou será utilizado o software, do mercado, dos riscos e de sua interação com outros sistemas de software e com o hardware. Ou seja, o contexto no qual o software em teste será utilizado no “mundo real” deve ser conhecido para, a partir daí, definir que casos de teste devem ser utilizados, com que prioridade e de que forma. Trata-se do teste dirigido ao contexto (*context driven testing*).

Das várias técnicas de teste dirigido ao contexto, a mais utilizada tem sido o teste de domínio (Kaner 2016). O teste de domínio inclui as técnicas de divisão do domínio em classes de equivalência e de análise de valores limite (Myers 1979) e, em geral, é utilizado seguindo-se a estratégia baseada em riscos. Algumas práticas gerais do teste de domínio são listadas a seguir:

- deve-se entender o domínio do software;
- dividi-lo em categorias (classes de equivalência, por exemplo);
- identificar elementos de cada categoria que possam ser utilizados para compor os casos de testes;
- priorizar os casos de teste;
- executar os casos de teste e avaliar os resultados gerados.

A fim de ajudar na priorização dos casos de teste, utiliza-se frequentemente a estratégia baseada em riscos, que tem o objetivo de selecionar os testes a serem realizados com base nos riscos do software e de seu contexto. Desta forma é possível focar o esforço do teste onde maior valor será agregado: em situações nas quais há maior probabilidade de encontrar defeitos e onde estes tenham um maior impacto no software.

Um risco pode ser definido como a possibilidade de haver problemas, danos ou perdas (Sommerville 2011). No teste de software, os riscos são considerados em três dimensões (Kaner et al 2002):

- a forma como o programa poderia falhar;
- com que probabilidade o programa poderia falhar desta forma;
- quais as consequências que esta falha poderia ter.

Para o teste de software, a dimensão mais importante é “a forma como o programa deveria falhar” (Kaner et al 2013). Por isto, a essência do teste baseado em riscos pode ser resumida nas seguintes ações:

- imagine como o produto poderia falhar;
- projete testes para expor estas falhas potenciais.

Imaginar como o programa pode falhar pode ser realizado de várias formas. Por exemplo, utilizando heurísticas para a identificação de riscos, junto com o conhecimento de domínio e a experiência dos testadores. Kaner e Bach (2004) consideram três classes de heurísticas para este fim:

- reconhecimento de sintomas do projeto que possam estar associados a riscos;
- aplicação do modo de falhas e análise de efeitos (*Failure Mode and Effects Analysis* – FMEA) para os elementos do produto e para os critérios mais importantes de qualidade do produto;
- aplicação de técnicas comuns (testes rápidos ou ataques) para identificar de forma rápida os tipos de erros mais comuns presentes no programa em teste.

Um exemplo de parte de resultado da análise de riscos é apresentado na Figura 1. Neste exemplo, os riscos são classificados pelo seu grau de exposição, e são relacionados à prioridade das características da qualidade. O grau de exposição do risco é calculado a partir da combinação entre o seu impacto e a probabilidade de ocorrência (Kaner e Bach 2004) (Sommerville 2011).

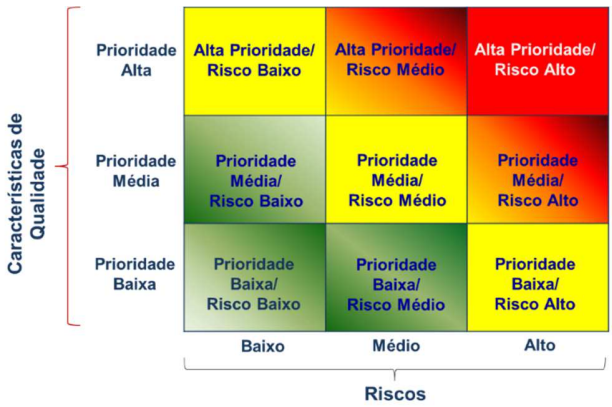

Figura 1. Ponderação Risco x Qualidade.

Técnicas de teste caixa-preta foram escolhidas para o tipo de curso referenciado neste artigo pelo fato de que os participantes do mesmo não têm conhecimento em desenvolvimento de software, mas sim do domínio e do contexto nos quais o software em teste será efetivamente utilizado.

3. ENSINO DE TESTE PARA PROFISSIONAIS QUE NÃO DESENVOLVEM SOFTWARE

O software é cada vez mais amplamente utilizado, adquirindo inclusive um papel fundamental para a realização de várias atividades críticas e complexas.

Profissionais que confiam no software para a execução de atividades críticas devem também saber testá-lo. Ainda que estes profissionais não sejam testadores, ensiná-los técnicas de teste de software aumenta o seu poder de análise e de crítica. Testar software possibilita uma investigação técnica qualificada, e esta investigação pode ser realizada de forma aprofundada caso o profissional saiba e possa utilizar seu conhecimento de domínio de forma direcionada, buscando a realização de testes eficientes (que revelem defeitos).

O teste baseado no contexto, no qual se insere o teste baseado no domínio, implementado através de técnicas de teste funcional, é baseado na aplicação de sete princípios básicos (Kaner et al 2002):

- o valor de qualquer prática depende do seu contexto;
- há boas práticas em um determinado contexto, mas não as melhores práticas;
- pessoas trabalhando juntas são a parte mais importante de qualquer contexto de projeto;
- projetos se desenrolam, ao longo do tempo, de formas não previsíveis;
- o produto é uma solução. Se o problema não foi resolvido, então o produto não funciona;
- bom teste de software é um processo intelectualmente desafiador;
- somente a partir de julgamento e habilidade, utilizados cooperativamente ao longo do projeto, é possível realizar as atividades certas, na hora certa, para efetivamente testar os produtos.

Por estes motivos, foi proposta uma disciplina na UFCSPA, para todos os cursos de graduação da Universidade, focados no domínio da Saúde, que ensina técnicas de teste dirigido ao contexto para profissionais que não têm conhecimento em desenvolvimento de software.

O domínio da saúde possui sua complexidade inerente. Primeiramente, por ser crítico e ser frequentemente utilizado em situações nas quais defeitos podem ter impacto em vidas humanas. E também por ser uma área com grande quantidade de conhecimento acumulado, normas e padrões estabelecidos.

A próxima seção apresenta os quatro padrões propostos a partir das experiências ensinando teste de software, no cenário apresentado.

4. PADRÕES PROPOSTOS

Para documentar os padrões definidos neste trabalho, foi utilizada a estrutura apresentada por James Coplien (1996). Estes padrões são apresentados nas tabelas a seguir.

Tabela 1 Padrão “Equilibrando Complexidade, Tempo e Utilidade em Cenários de Teste”.

NOME	Equilibrando Complexidade, Tempo e Utilidade em Cenários de Teste
PROBLEMA	Para ensinar técnicas de teste de software caixa-preta, é importante caracterizar contextos através da especificação de cenários nos quais o teste será realizado. Estes cenários devem ser simples o suficiente para serem utilizados durante as aulas, e complexos o suficiente para mostrarem as características das práticas e técnicas de teste. Ao mesmo tempo, devem ser representativos o suficiente para que o aluno facilmente possa escalá-los a problemas reais. **Como definir cenários de teste simples o suficiente para serem utilizados em aula e complexos o suficiente para facilmente serem escalados a problemas reais?**
CONTEXTO	Cursos de técnicas de teste caixa-preta para profissionais sem conhecimento em desenvolvimento de software têm a duração de 30 horas. Aproximadamente 10 horas são utilizadas para a apresentação de conceitos sobre teste de software. Nas outras 20 horas, são realizados exercícios práticos aplicando os conceitos utilizados, a partir de contextos compostos por cenários de teste definidos a partir de software da área da saúde.
EXEMPLO MOTIVACIONAL	Alguns exemplos frequentemente utilizados na literatura e em cursos sobre técnicas de teste caixa-preta clássicas tais como particionamento em classes de equivalência e análise de valores limite são o de um programa que soma dois números inteiros e o de um programa que recebe três medidas de lados de um triângulo e gera como saída o tipo do triângulo (isósceles, escaleno ou

	equilátero). Profissionais da área da saúde que estão aprendendo estas técnicas dificilmente conseguem transpor os objetivos e a dinâmica de utilização das técnicas a partir destes exemplos simples para o contexto de sistemas de software críticos e complexos do domínio da saúde. Alguns exemplos: como aplicar as técnicas para um software realiza a análise de um mapeamento genético ou de um software que auxilia na anamnese em deficiências sensoriais, voltada a neurologia infantil? Por outro lado, professores ou instrutores que ensinam as técnicas de teste normalmente não possuem este conhecimento avançado em domínios tão específicos e complexos.
FORÇAS	É bastante frequente, na literatura, o uso de cenários de teste bastante simples. Ainda que sejam didáticos, muitas vezes a simplicidade acaba passando ao aluno a ideia de que as técnicas são também muito simples, não gerando valor em cenários da vida real. Por outro lado, se cenários de um contexto real de aplicação são utilizados, a complexidade inerente à situação torna a aplicação da técnica de teste recém aprendida muito difícil, o que também pode dar ao aluno a impressão de que as técnicas de teste não são aplicáveis em cenários da vida real. Também é importante considerar que as técnicas devem ser ensinadas de forma que os alunos não só aprendam a utilizá-las, mas também desenvolvam a capacidade para entender quando estas podem ser aplicadas com suas debilidades e seus pontos fortes, assim como para adaptá-las, caso necessário, para situações específicas, sem perder o valor da sua essência nem ir contra os princípios que nortearam a proposta da técnica.
SOLUÇÃO	**Aproveite o fato de os alunos terem conhecimento de domínio e faça-os escolher o software e as funcionalidades que serão submetidas ao teste.** Como os alunos do curso são profissionais que têm conhecimento de domínio de uma ou mais áreas da saúde, e já utilizam alguns sistemas de software deste domínio, solicita-se, já no início do curso, que estes próprios alunos selecionem o software que será testado. O software pode ser selecionado por duplas de alunos, ou individualmente, deve ser da área da saúde. A partir da escolha do software, os alunos identificam um conjunto restrito de funcionalidades que serão utilizadas para a aplicação das técnicas de teste apresentadas. O professor tem a responsabilidade de revisar a escolha do software, a identificação do subconjunto de funcionalidades e a aplicação das técnicas, com o cuidado de equilibrar a complexidade a partir da proposta dos alunos. Caso a aplicação das técnicas seja realizada de forma muito simplificada, ou esteja sendo dificultada, o professor deve sugerir a ampliação ou a redução do escopo de teste.
EXEMPLO RESOLVIDO	O aprofundamento no conhecimento das técnicas de teste caixa-preta e sua aplicação em exemplos de software fornecidos pelo próprio proporcionam uma interação de conhecimento entre professores/instrutores e alunos. Por um lado, os professores/instrutores conhecem bem as técnicas de teste, incluindo suas características relacionadas à análise de domínio. Por outro lado, os alunos conhecem bem o domínio e o contexto de uso do software em teste. A integração destes conhecimentos faz com que seja possível a utilização efetiva de técnicas de teste para a avaliação de atributos de sistemas de software específicos da área da saúde, gerando valor para todos os envolvidos.
CONTEXTO RESULTANTE	Como a escolha do software a ser testado, para a aplicação das técnicas de teste propostas no curso, é realizada pelos próprios alunos, estes tendem a escolher sistemas de software nos quais têm interesse. Este é um fator de motivação para a realização dos testes, que ajudam nesta investigação das funcionalidades do software. Outro fator de motivação é que as funcionalidades do escopo de teste também são escolhidas pelo aluno. Assim este é envolvido na problematização da situação. Caso haja problemas na definição do escopo de teste, seja por este ser simples ou complexo demais, há intervenção do professor, que deve realizar avaliações graduais do conteúdo apresentado. Os alunos tendem a entender a intervenção do professor, já que estão realizando os testes de forma ativa, e não apenas aprendendo as técnicas de maneira conceitual.
RACIONAL	A aplicação das técnicas de teste em um contexto real, que tenha significado ao aluno, é fundamental para que este aluno desenvolva capacidade e competência para a realização dos testes (Kaner et al 2013). O envolvimento do aluno na seleção do software em teste e nas eventuais mudanças de escopo de teste, a partir da experiência da aplicação das técnicas, é direto, acontece de forma ativa, consciente e intencional, conforme proposto nas metodologias ativas de ensino-aprendizagem (Barell 2007), que têm sido bastante discutidas e aplicadas na formação de competências de profissionais de várias áreas de atuação.

Tabela 2 Padrão “Não existe teste perfeito!”.

NOME	Não existe teste perfeito!
PROBLEMA	É impossível testar todas as combinações possíveis de um determinado cenário de teste. Testa-se sempre parcialmente o escopo de teste. Além disso, mesmo que defeitos não sejam mais encontrados depois de um certo tempo testando, não é possível afirmar que o software esteja correto. Esta definição frequentemente desmotiva o testador, já que, não importa o quão bom ele seja, o teste sempre estará, de certa forma, incompleto. O aluno deve entender este conceito e utilizá-lo para refinar a aplicação das técnicas de teste. **Como não desmotivar os alunos com a impossibilidade de teste exaustivo?**
CONTEXTO	Nas primeiras aulas do curso de técnicas de teste caixa-preta, os princípios de teste são apresentados (ISTQB 2011), incluindo: “teste mostra a presença de defeitos” e “teste exaustivo é impossível”. Estes conceitos fazem parte das aulas teóricas, e devem ser considerados nos momentos de aplicação das técnicas de teste no sistema de software escolhido, sem que estes pontos se transformem em uma frustração antecipada com as atividades.
EXEMPLO MOTIVACIONAL	Quando alunos da área da saúde recebem a informação sobre a impossibilidade de teste exaustivo do software, em geral manifestam preocupação pela vulnerabilidade do processo. Principalmente pela criticidade da área, na qual falhas de software podem ter alto impacto. Deparando-se com este panorama, alguns alunos não entendem como as técnicas de teste podem ajuda-los, devido a esta limitação. E insistem ou em tentar realmente realizar o teste exaustivo, ou então não investem o esforço e atenção necessários para selecionar casos de teste confiáveis (que revelam defeitos). Ambas as situações são prejudiciais ao processo de teste de software, e levam a uma situação na qual os resultados obtidos são restritos e não geram o valor que poderiam e deveriam para a avaliação do software.
FORÇAS	As debilidades decorrentes da aplicação das técnicas de teste caixa-preta devem ser apresentadas e discutidas em sala de aula, pois entende-las e lidar com elas deve fazer parte das habilidades e competências de um testador. Mais do que isto, estas debilidades devem ser consideradas na escolha e na aplicação das técnicas de teste, balanceando custo e benefício em cada etapa. A estratégia de teste baseada em riscos é um exemplo de aplicação destes conceitos para refinar os testes. Ao mesmo tempo, conhece-las pode levar à frustração e à crença de que não importa o quão sistemático e criterioso seja o teste, este será sempre incompleto. Então, em vez de “perder tempo” planejando as atividades de teste, pode parecer que o melhor seja realiza-las de qualquer forma. Esta situação também dificulta a realização de atividades de reteste e de teste de regressão, principalmente quando os casos de teste executados não foram documentados como deveriam.
SOLUÇÃO	**Apresente um *template* de plano de testes, para ser utilizado no processo de teste do software escolhido e faça-os revisar e avaliar a qualidade dos testes realizados por colegas, considerando o escopo por eles definido no plano de testes.** Uma vez que o escopo de teste (funcionalidades do sistema de software escolhido) seja definido, propõe-se aos alunos um *template* de documentação dos casos de teste planejados e dos casos executados, junto com os resultados encontrados. Este *template* deve ser bastante simples. Se possível, deve ser documentado em uma ferramenta de gerência de teste. Ensina-se ao aluno a importância de documentar o que foi testado, para explicitar o escopo do teste. Assim qualquer defeito que por ventura seja encontrado fora do escopo do teste realizado não deve ser visto como uma falha do processo de teste. Ao contrário de defeitos que sejam encontrados nas funcionalidades testadas. Mostra-se ao aluno que a documentação dos testes tem vários objetivos: • facilitar a realização de retestes e de testes de regressão; • avaliar a qualidade e a extensão dos testes realizados; • documentar a extensão dos testes realizados, ou seja, a cobertura das funcionalidades testadas, a fim de balancear a garantia aos testes realizados. Outra prática interessante é pedir ao aluno que revise e avalie a extensão e a qualidade dos testes realizados por outros alunos. Esta avaliação é feita revisando-se os casos de teste documentados, analisando-se de forma exploratória os testes e finalizando com uma reunião com os testadores, para que estes possam explicar suas decisões com relação ao escopo de funcionalidades testadas.
EXEMPLO RESOLVIDO	Uma vez que os alunos entendem que não é um problema não poder executar todos os casos de teste possíveis para um determinado contexto, eles conseguem ver o valor do planejamento e da documentação dos testes, entendendo também o valor que este processo agrega ao entendimento e à atribuição de confiabilidade ao software.
CONTEXTO RESULTANTE	Como a escolha do software a ser testado e a documentação dos casos de teste planejados e executados, o aluno tem a noção prática dos custos e benefícios dos testes realizados, entendendo

	que, mesmo que “tente” realizar teste exaustivo, o custo será tão alto que não valerá a pena, considerando a quantidade provável de defeitos extras que poderia encontrar. Além disso, revisando e avaliando o trabalho de seus pares, o aluno tem a oportunidade de analisar outras formas de definição de escopo de testes, considerando as decisões, ameaças e ganhos nestes cenários.
RACIONAL	O entendimento de princípios do teste (ISTQB 2011) é fundamental para que o aluno tenha uma visão realista e completa das atividades de teste. Além disso, seu envolvimento direto, tanto na documentação dos casos de teste, como na revisão dos casos de teste de seus pares faz com que, assim como no padrão apresentado na Tabela 1, o aluno tenha um papel ativo na aprendizagem, não somente assimilando conceitos, mas utilizando-os na prática, em contextos escolhidos por ele.

Tabela 3 Padrão “Não existe mundo perfeito para o teste!”.

NOME	Não existe mundo perfeito para o teste!
PROBLEMA	É muito comum que as atividades de teste tenham que ser realizadas sob pressão, com fortes restrições de custos e de prazo. As razões para esta situação são culturais, devido à maior valorização de atividades de análise, projeto e construção de software do que de teste de software, devido à rara utilização do teste sistemático e à crença de que “qualquer um pode testar software”. Mesmo com restrições fortes, o teste deve ser realizado e documentado. Entretanto, os alunos frequentemente percebem que, nesta situação, o melhor é executar os testes de forma não sistemática, já que assim “perderão” menos tempo planejando e documentando, “ganhando” mais tempo executando efetivamente os testes. **Como mostrar aos alunos a importância de planejar e documentar os testes, mesmo em projetos com fortes restrições de orçamento e de prazos?**
CONTEXTO	É importante apresentar contextos reais nas aulas do curso de técnicas de teste caixa-preta. Assim tem-se mais condições de formar testadores que efetivamente realizam os testes após o curso, entendendo sua importância e seus objetivos. Por isto, é importante mostra que situações restritivas existem, e que justamente nestas situações a importância da realização de teste sistemático aumenta, pois é importante que os poucos recursos disponíveis sejam utilizados para gerar o maior valor possível com o teste.
EXEMPLO MOTIVACIONAL	Alunos da área da saúde devem estar conscientes da importância e do impacto do software neste contexto. Neste sentido, funcionalidade e confiabilidade são características fundamentais. Quando se explica a estes alunos o conceito de priorização, o fato de itens com prioridade mais baixa não serem considerados acaba chamando mais atenção do que a consideração de itens com prioridade mais alta. E, com a ideia de que a área da saúde é crítica e pode implicar em prejuízo de vidas humanas, simplesmente desconsiderar casos de teste com menor prioridade causa sentimento de insegurança nestes profissionais.
FORÇAS	As debilidades decorrentes da aplicação das técnicas de teste caixa-preta devem ser apresentadas e discutidas em sala de aula, pois entende-las e lidar com elas deve fazer parte das habilidades e competências de um testador. Mais do que isto, estas debilidades devem ser consideradas na escolha e na aplicação das técnicas de teste, balanceando custo e benefício em cada etapa. A estratégia de teste baseada em riscos é um exemplo de aplicação destes conceitos para refinar os testes. Ao mesmo tempo, conhece-las pode levar à frustração e à crença de que não importa o quão sistemático e criterioso seja o teste, este será sempre incompleto. Então, em vez de “perder tempo” planejando as atividades de teste, pode parecer que o melhor seja realiza-las de qualquer forma. Se em situações com fortes restrições o teste sistemático não é utilizado, dificilmente os testadores terão condições de mostrar a seus gerentes ou aos seus superiores a importância da realização dos testes e as consequências do teste realizado com fortes restrições.
SOLUÇÃO	**Ensine o aluno a fazer o planejamento dos testes utilizando níveis e formas de cobertura, além de fazê-lo avaliar projetos de colegas.** Propõe-se ao aluno que defina um plano de testes, no qual cada funcionalidade é apresentada com os seguintes atributos: descrição geral, prioridade e cobertura de teste planejado. A cobertura dos testes é definida através de níveis de teste: • Nível 1: teste do “caminho feliz” da funcionalidade. Por exemplo, para a funcionalidade de *login*, o “caminho feliz” é executado quando o *login* é realizado com sucesso com dados de um usuário já cadastrado no sistema;

	• Nível 2: teste envolvendo condições negativas, valores limite e fluxos de caminhos alternativos. No caso da funcionalidade de *login*, um exemplo de caminho alternativo seria tentar realizado o *login* com os dados de um usuário que já foi excluído do cadastro do sistema; • Nível 3: teste envolvendo práticas gerais com a utilização de outros idiomas ou com diferentes plataformas (se for o caso). O planejamento de que níveis devem ser testados para cada funcionalidade é feito com base nos riscos e na análise de negócio. Embora qualquer combinação de prioridades e de níveis de teste possa ser adequada, dependendo dos objetivos de teste, dois exemplos mais comuns de estratégias relacionadas ao planejamento dos níveis de teste são apresentados a seguir: • **Cobertura por amplitude**, na qual busca-se executar a maior quantidade possível de funcionalidades, seguindo a ordem de prioridade das mesmas, considerando o nível 1 de teste. Caso ainda haja tempo, busca-se executar o nível 2 de teste das mesmas funcionalidades e, da mesma forma, caso ainda haja tempo, o nível 3 é executado. Esta estratégia pode ser diretamente comparada ao *smoke testing* (teste de fumaça ou teste rápido). O objetivo é realizar testes preliminares no sistema de forma a identificar falhas simples, mas que sejam críticas. Enquanto os defeitos que causaram este tipo de falha estiverem presentes no software, não há motivos para serem realizados testes mais elaborados (Kaner 2002); • **Cobertura por profundidade**: busca-se executar os três níveis de teste para as funcionalidades com prioridade 1. Posteriormente, caso haja tempo, são executados os casos de teste para os níveis 1 e 2, das funcionalidades com prioridade 2. E, caso ainda haja tempo, executa-se os testes para as funcionalidades com prioridade 3, considerando o primeiro nível de testes. Outra prática interessante é pedir ao aluno que revise e avalie o plano de testes realizados por outros alunos, juntamente com o entendimento do contexto do software testado por seus pares.
EXEMPLO RESOLVIDO	A decisão de considerar a priorização com base na relação de risco e valor agregado diminui a preocupação dos alunos, pois estes entendem que as situações mais críticas, que podem causar prejuízo em vidas humanas, caso estejam associadas a falhas, são cobertas e priorizadas no planejamento e na execução dos testes.
CONTEXTO RESULTANTE	Como a escolha do software a ser testado e a documentação do plano de teste, analisando a prioridade de cada funcionalidade, com base nos riscos associados, o aluno tem a noção prática dos custos e benefícios dos testes realizados, entendendo que os testes devem ser priorizados, e que qualquer escolha de casos de teste a ser executado leva também à escolha de casos que não serão executados. Além disso, revisando e avaliando o trabalho de seus pares, o aluno tem a oportunidade de analisar outras formas de priorização dos testes, considerando as decisões, ameaças e ganhos nestes cenários.
RACIONAL	O entendimento de como realizar os testes sob restrições é fundamental para que o aluno tenha uma visão realista e completa das atividades de teste. Além disso, seu envolvimento direto e sua responsabilidade na definição e documentação do plano de testes e na revisão do plano de teste de seus pares faz com que, assim como nos padrões apresentados na Tabela 1 e na Tabela 2, o aluno tenha um papel ativo na aprendizagem.

Tabela 4 Padrão “Defeitos Similares em Situações Similares”.

NOME	Defeitos Similares em Situações Similares
PROBLEMA	Mesmo utilizando a estratégia de testes baseada em riscos, é difícil para pessoas que ainda não são testadores experientes identificar grande quantidade de defeitos de forma rápida e eficiente, mesmo utilizando as técnicas de teste caixa-preta. Ao mesmo tempo, sistemas de software similares, utilizados em um mesmo domínio, ou então funcionalidades similares, mesmo em sistemas e domínios distintos, tendem a se repetir. Este conhecimento poderia ajudar na priorização de funcionalidades que tenham maior probabilidade de apresentarem defeitos. **Como mostrar a profissionais que não são desenvolvedores, nem testadores experientes, a ideia de *clusters* de defeitos?**
CONTEXTO	Cursos de técnicas de teste caixa-preta para profissionais sem conhecimento em desenvolvimento de software têm a duração de 30 horas. Aproximadamente 10 horas são utilizadas para a apresentação de conceitos sobre teste de software. Nas outras 20 horas, são realizados exercícios práticos aplicando os conceitos utilizados, a partir de contextos compostos por cenários de teste definidos a partir de software da área da saúde. É importante utilizar o máximo possível os recursos disponíveis em um determinado domínio (área da saúde), para um determinado tipo de software ou determinado tipo de funcionalidade, para otimizar os testes, para encontrar mais defeitos em menos tempo e com menor esforço.

EXEMPLO MOTIVACIONAL	Quando alunos da área da saúde conhecem o conceito de “catálogo de defeitos”, todos os defeitos registrados parecem bastante compatíveis com o software deste domínio. Neste contexto, a ideia de que “quando tudo é importante, nada é importante” é bastante aplicável, já que o aluno tem o impulso de tentar definir casos de teste para todos os defeitos apresentados nos catálogos.
FORÇAS	Defeitos são causados por vários motivos, tais como a complexidade de funcionalidades, do domínio ou do contexto de uso do software. Em situações similares, há uma maior probabilidade de que defeitos similares sejam encontrados. Caso estes defeitos sejam utilizados como base para a definição dos casos de teste a serem executados, junto com os critérios apresentados pelas técnicas de teste de software, tem-se a tendência de realizar testes mais robustos, com maior probabilidade de encontrar defeitos.
SOLUÇÃO	**Oriente a construção de um catálogo de defeitos que ajude o aluno a entender o comportamento dos defeitos e a forma como estes se agrupam no software.** Como os alunos do curso são profissionais que têm conhecimento de domínio de uma ou mais áreas da saúde, e já utilizam alguns sistemas de software deste domínio, solicita-se, logo após a seleção do software que será testado, o início da construção de um “Catálogo de Defeitos” conjunto, da turma, via *wiki*. Este catálogo é construído de forma colaborativa, contendo informações sobre defeitos encontrados, juntamente com a descrição de seu contexto e caso de teste utilizado. Esta é uma abordagem utilizada por Kaner (1998), entre outros autores. Algumas formas de utilização das informações deste catálogo são listadas a seguir: • geração de casos de teste, comparando-se os defeitos encontrados com a percepção de riscos que existe sobre o programa; • identificação de tipos de defeitos que normalmente não seriam considerados; • treinamento de pessoas nas atividades de teste. O professor tem a responsabilidade de revisar continuamente o catálogo de defeitos, de forma a garantir sua atualização, confiabilidade e utilidade.
EXEMPLO RESOLVIDO	Analisando os tipos de defeitos mais comuns na área da saúde, e as causas mais comuns para estes defeitos (a partir de uma análise funcional), os alunos adquirem uma maior confiança para focar seu esforço na definição de casos de teste que os identifiquem, não desperdiçando esforço em defeitos pouco comuns neste contexto.
CONTEXTO RESULTANTE	A construção do catálogo de defeitos, de forma colaborativa, é mais uma ação para o envolvimento ativo dos alunos no entendimento do software sob o ponto de vista de realização dos testes. Trata-se de uma forma de auxiliá-los a desenvolver habilidades para pensar e agir como testadores. Além disso, trata-se de uma forma de compartilhamento de informações sobre defeitos, envolvendo diferentes sistemas de software, de diferentes subdomínios da área da saúde. Pretende-se desenvolver no aluno a capacidade de entender as similaridades entre as várias categorias de funcionalidades, compreendendo que os planos de teste devem ser desenvolvidos a partir de conhecimento já adquirido, mesmo este conhecimento tenha sido desenvolvido com o teste de outros sistemas de software.
RACIONAL	A atividade de construção do catálogo geral de defeitos permite a aplicação do modo de falhas e análise de efeitos (*Failure Mode and Effects Analysis* – FMEA) para os elementos do produto e para os critérios mais importantes de qualidade do produto. Pensar no modo *failure* é pensar na forma como o programa pode falhar. Ou seja, buscar riscos que possam ser utilizados para a determinação de casos de teste. Para cada potencial falha, deve-se perguntar: • como é esta falha? • como a falha poderia ser identificada? • o quão caro seria buscar a falha? • quem ser impactado pela falha? • quanta variação poderia ocorrer no efeito da falha? • o quão séria poderia ser a falha? • o quão caro seria corrigir a causa desta falha? Com base nesta análise, decide-se se há benefícios em seguir buscando pela falha potencial, através de novos casos de teste. Novamente, como nos outros padrões apresentados, cita-se o envolvimento ativo do aluno em uma atividade intimamente relacionada ao teste de software, como uma ferramenta para acelerar e consolidar sua aprendizagem em técnicas de teste de caixa-preta.

5. CONSIDERAÇÕES FINAIS

Um conjunto de quatro padrões extraídos a partir da experiência do ensino de teste para profissionais sem conhecimento em desenvolvimento de software foi apresentado. A experiência mencionada neste trabalho foi obtida a partir da realização de cursos de técnicas de teste caixa-preta em uma Universidade com cursos de graduação focados na área da saúde.

Pretende-se realizar várias edições deste curso, refinando o conteúdo, as atividades e as abordagens utilizadas, para melhor cumprir seu objetivo. Estas várias edições permitirão a realização de análises mais aprofundadas a partir de resultados obtidos, corroborando ou não as soluções documentadas nos padrões.

Também pretende-se definir novos padrões, com base em teorias de ensino-aprendizagem, em teorias de ensino de técnicas de teste e em adaptações das técnicas de teste existentes para a avaliação mais aprofundada de funcionalidades de software da área da saúde.

REFERÊNCIAS BIBLIOGRÁFICAS

Barell, J. 2007. *Problem-based learning: An inquiry approach*. Corwin Press.

Bergin, J. 2012. *Pedagogical Patterns: Advice for Educators*. Joseph Bergin Software Tools, 2012. 190 p.

Coplien, J. 1996. *Software Patterns*. SIGS Books, New York.

ISTQB. 2011. *Certified Tester Foundation Level Syllabus*. Disponível em: http://www.istqb.org/downloads/viewdownload/16/15.html.

Kaner, C.; Falk, J.; Nguyen, H. Q. 1998. *Testing Computer Software*, 2. ed. New York, NY – Estados Unidos. John Wiley & Sons. 480 p.

Kaner, C; Bach, J.; Pettichord, B. 2002. *Lessons Learned in Software Testing*, John Wiley & Sons.

Kaner, C; Bach, J. 2004. *The nature of exploratory testing*. Tampere, Finland. Disponível em: http://kaner.com/pdfs/NatureOfExploratoryTest.pdf

Kaner, C. 2009. *Software Testing as a Quality Improvement Activity*. Lockheed Martin / IEEE Computer Society Webinar Series. Disponível em: http://kaner.com/pdfs/TESTINGkanerIEEE2009.pdf.

Kaner C. et al. 2013. *The Domain Testing Workbook*, Context Driven Press.

Kaner, C. 2016. *Updating BBST to Version 4.0*. Disponível em: http://kaner.com/.

Laurillard, D. 2012. *Teaching as a Design Science: Building Pedagogical Patterns for Learning and Technology*. Routledge. 272 p.

Marick, B. 1995. *The Craft of Software Testing: Subsystem Testing Including Object-Based and Object-Oriented Testing*. Prentice Hall.

Marick, B. 2016. *Some Test Patterns*. Disponível em: http://www.exampler.com/testing-com/test-patterns/patterns/index.html.

Myers, G. 1979. *The Art of Software Testing*. New York: John Willey & Sons. 170 p.

Sommerville, I. 2011. *Engenharia de Software*. 9. ed. Pearson Prentice-Hall. 529 p.

The Hillside Group. 2016. *Pattern Languages of Programs (PLOP) Conferences*. Disponível em: http://www.exampler.com/testing-com/test-patterns/patterns/index.html

UFCSPA. 2016. *Site institucional*. http://www.ufcspa.edu.br/.

UFCSPA. 2016. *Site do curso de graduação da Informática Biomédica – Perfil do Aluno*. http://www.ufcspa.edu.br/index.php/cursos/informatica-biomedica/perfil-do-aluno