Atividade Semana 06 - Testes Exploratórios

SQUAD 1: Andrey Felipe de Lima; Daniela Siana Pabis; Eduardo Neves de Souza; João Pedro de Jesus Perin.

O que são testes exploratórios?

O Teste Exploratório é uma abordagem de teste de software centrada na liberdade e na expertise do testador. Diferente das metodologias tradicionais, que dependem de roteiros e scripts pré-definidos, nesta modalidade o aprendizado, o design dos testes e a sua execução ocorrem de forma simultânea.

Em vez de apenas verificar se o sistema responde a comandos específicos, o testador "explora" a aplicação como um usuário real faria, mas com o olhar crítico de um especialista. O objetivo principal é a descoberta: encontrar defeitos complexos, comportamentos inesperados e problemas de usabilidade que o planejamento formal e os testes automatizados muitas vezes não conseguem prever.

Pilares da Abordagem:

Para entender como essa prática funciona no dia a dia, podemos destacar três pontos fundamentais:

- Intuição e Instinto: Como defendido por Cem Kaner, a prática valoriza o "sentimento" do testador. Quando os casos de teste formais se esgotam, o profissional utiliza sua experiência para investigar áreas nebulosas do código;
- Simulação de Uso Real: Ao não se prender a um "passo a passo" rígido, o teste exploratório consegue simular cenários dinâmicos e fluxos de trabalho imprevisíveis, aproximando-se da experiência de uso no mundo real;
- Agilidade na Investigação: Por não exigir uma preparação extensiva de documentação antes da execução, ele permite que defeitos críticos sejam identificados e comunicados rapidamente, sendo um aliado essencial em estratégias modernas de Garantia de Qualidade (QA).

Nota Histórica: Embora tenha raízes no que antes chamávamos de "testes ad-hoc", o teste exploratório evoluiu para uma disciplina respeitada e estruturada, sendo hoje um componente vital para uma cobertura de testes abrangente e eficaz.

Por que são usados?

Os testes exploratórios podem ser classificados como um complemento fundamental aos testes baseados em abordagens estruturadas. Segundo Yu et al. (2021, p. 44), “Alguns bugs não podem ser descobertos por testes automatizados, mas podem ser detectados por testes exploratórios [...]”. Esta fala reforça a relevância da abordagem na identificação de defeitos que não seriam facilmente detectados por métodos tradicionais.

Nesse sentido, os testes exploratórios são utilizados para ampliar a abrangência (cobertura) de testes, permitindo a avaliação de cenários imprevistos, casos extremos e comportamentos inesperados do sistema. Essa abordagem favorece a identificação precoce de defeitos, o aprendizado contínuo sobre a aplicação e a melhoria da experiência do usuário, atuando como um importante apoio aos testes estruturados e automatizados no processo de garantia da qualidade do software.

Demais vantagens e contextos favoráveis podem ser descritos, tais como:

- Ambientes com requisitos rasamente definidos e/ou que estejam em constante mudança;
- Descobrir novos cenários e expandir a gama de testes;
- Velocidade quanto ao respaldo de feedbacks sobre a qualidade do produto, sobretudo em ciclos de desenvolvimento curto;
- aproveitamento e aprimoramento do conhecimento, da intuição e da experiência do testador no processo de teste.

Quais técnicas de testes exploratórios existem?

Sessões Baseadas em Charter (Charter-Based Testing)

Definição:

Sessões Baseadas em Charter são uma técnica de teste exploratório estruturada, na qual cada sessão de teste é orientada por um charter, isto é, um objetivo de exploração previamente definido. O charter estabelece o que deve ser explorado, mas não determina como testar, dispensando a criação de casos de teste formais e permitindo maior flexibilidade ao testador.

Como Executar:

1. O time de testes define um charter claro, especificando escopo, foco e objetivo da sessão;
2. É estabelecido um tempo fixo (time-box) para a execução, geralmente entre 30 e 90 minutos;
3. O testador executa os testes de forma livre, porém alinhado ao objetivo definido no charter;

4. Durante a sessão, são registrados defeitos encontrados, riscos identificados, dúvidas e observações relevantes;
5. Ao final da sessão, o testador elabora um relatório da sessão, consolidando os resultados obtidos.

O que fazer durante a execução:

- Verificar erros de navegação e inconsistências no fluxo do sistema;
- Explorar tanto os fluxos principais quanto os fluxos alternativos;
- Forçar a entrada de dados inválidos, incompletos ou extremos;
- Analisar mensagens exibidas ao usuário e a correta aplicação das regras de negócio;
- Observar comportamentos inesperados ou não documentados.

Pontos fortes:

- Mantém um equilíbrio entre estrutura e liberdade;
- Facilita a documentação dos testes exploratórios e seus resultados;
- Favorece o aprendizado contínuo sobre o sistema;
- Direciona o esforço de teste sem engessar a execução;
- Muito utilizada em ambientes ágeis.

Limitações:

- Depende fortemente da capacidade analítica e experiência do testador;
- Pode deixar lacunas de cobertura caso os charters sejam mal definidos.

Teste Ad-hoc

Definição:

Teste Ad-hoc é uma técnica de teste exploratório não estruturada, realizada sem planejamento prévio ou definição de objetivos formais. Baseia-se exclusivamente na experiência, intuição e criatividade do testador, que interage livremente com o sistema.

Como Executar:

1. O testador acessa o sistema sem qualquer roteiro ou planejamento formal;
2. Executa ações aleatórias, não convencionais ou fora do fluxo esperado;
3. Observa as reações do sistema frente a essas interações;
4. Registra os defeitos encontrados.

Observação importante: sem registros adequados, pode ser difícil reproduzir os defeitos identificados.

O que fazer durante a execução:

- Analisar quebras inesperadas do sistema;
- Gerar erros não tratados automaticamente;
- Provocar travamentos;
- Identificar comportamentos inconsistentes;
- Executar ações fora da ordem lógica;
- Inserir dados inválidos ou extremos;
- Interromper processos em execução.

Pontos fortes:

- Rápido de executar;
- Exige pouca preparação;
- Útil para descobertas iniciais de defeitos.

Limitações:

- Difícil rastreabilidade;
- Baixa repetibilidade;
- Alto risco de esquecimento dos passos executados.

Tour Testing (Test Tours)

Definição:

Tour Testing é uma técnica exploratória que organiza a exploração do sistema em “tours”, cada um com um foco específico, simulando uma visita guiada ao sistema sob diferentes perspectivas.

Exemplos de Tours:

- Tour do Usuário Iniciante: avalia a primeira experiência de uso;
- Tour de Erros: foca em provocar falhas;
- Tour de Configuração: explora opções e preferências;
- Tour de Dados: testa limites, campos vazios e dados extremos.

Como Executar:

1. Define-se previamente o tipo de tour a ser realizado;
2. O sistema é explorado sob aquele ponto de vista específico;
3. São registradas dificuldades, defeitos e riscos encontrados.

O que fazer durante a execução:

- Avaliar a experiência do usuário;
- Testar fluxos críticos;
- Analisar feedbacks e mensagens exibidas pelo sistema.

Pontos fortes:

- Organiza o teste exploratório;
- Excelente para avaliação de usabilidade;
- Fácil adaptação a diferentes contextos.

Limitações:

- Pode não cobrir regras internas profundas;
- Depende da criatividade e experiência do testador.

Teste Baseado em Riscos

Definição:

Técnica que prioriza a exploração das funcionalidades que apresentam maior risco ao negócio, ao usuário ou à segurança, concentrando o esforço de teste nas áreas mais críticas.

Como Executar:

1. Identificar as áreas críticas do sistema;
2. Avaliar impacto e probabilidade de falha;
3. Executar testes intensivos nessas áreas prioritárias.

O que fazer durante a execução:

- Observar dificuldades de uso;
- Analisar fluxos confusos e erros não explicados;
- Verificar ausência de feedback ao usuário;
- Explorar cenários críticos;
- Forçar falhas;
- Avaliar impactos graves.

Pontos fortes:

- Otimiza o tempo de teste;
- Foca no que é mais importante;
- Muito eficiente em prazos curtos.

Limitações:

- Funcionalidades de baixo risco podem não ser testadas;
- Requer bom entendimento do negócio.

Teste por Heurísticas

Definição:

Técnica que utiliza heurísticas, ou regras práticas, como guias mentais para orientar a exploração do sistema.

Heurísticas comuns:

- CRUD: Create, Read, Update, Delete;
- Entradas inválidas: campos vazios, caracteres especiais, valores extremos;
- Estados do sistema: antes, durante e depois das ações;
- Mensagens: clareza, utilidade e orientação ao usuário.

Observação importante: Heurísticas não são regras fixas, mas atalhos de pensamento.

Como funciona:

1. Selecionar heurísticas adequadas ao sistema;
2. Aplicá-las durante a exploração;
3. Registrar falhas e inconsistências encontradas.

O que fazer durante a execução:

- Verificar ausência de validações;
- Analisar erros repetitivos e inconsistências entre telas;
- Testar entradas inválidas;
- Avaliar estados do sistema.

Pontos fortes:

- Rápida aplicação;
- Alta eficiência;
- Boa cobertura lógica.

Limitações:

- Exige experiência;
- Pode deixar lacunas se poucas heurísticas forem utilizadas.

Teste Baseado em Personas

Definição:

Técnica que simula o uso do sistema por diferentes perfis de usuários, reais ou fictícios, chamados de personas.

Como Executar:

1. Definir as personas relevantes;
2. Executar os fluxos sob cada perfil;
3. Analisar dificuldades e problemas encontrados.

O que fazer durante a execução:

- Verificar dificuldades de entendimento;
- Analisar erros comuns do usuário;
- Avaliar ausência de ajuda ou feedback;
- Identificar problemas de acessibilidade;
- Simular erros humanos;
- Avaliar clareza do sistema.

Pontos fortes:

- Forte foco no usuário final;
- Melhora a experiência do usuário (UX);
- Detecta falhas de entendimento.

Limitações:

- Menor foco em regras técnicas;
- Depende de boas definições de persona.

Teste de Aprendizado Progressivo

Definição:

Abordagem em que os testes evoluem conforme o aprendizado contínuo do testador sobre o sistema.

Como Executar:

1. Realizar exploração inicial.
2. Aprender regras e comportamentos do sistema.
3. Ajustar a estratégia de teste.
4. Realizar nova exploração.

Trata-se de um ciclo contínuo:

- explorar → aprender → ajustar → explorar novamente

O que fazer durante a execução:

- Refinar regras implícitas;
- Identificar dependências ocultas;
- Analisar comportamentos condicionais;
- Refinar hipóteses;
- Explorar exceções.

Pontos fortes:

- Exploração profunda;
- Excelente para sistemas novos.

Limitações:

- Difícil estimar esforço;
- Depende da experiência do testador.

Teste de Ataque (Error Guessing)

Definição:

Técnica baseada na experiência prévia do testador, focada em prever pontos onde o sistema tende a falhar.

Como Executar:

1. Analisar falhas recorrentes em sistemas semelhantes;
2. Criar cenários de erro;
3. Executar testes direcionados.

O que fazer durante a execução:

- Verificar falhas silenciosas;
- Analisar comportamentos não tratados;
- Testar cenários conhecidos de falha;
- Forçar exceções;
- Avaliar estabilidade.

Pontos fortes:

- Muito eficiente;

- Alta taxa de defeitos encontrados.

Limitações:

- Forte dependência da experiência;
- Pouco eficaz para iniciantes.

Exemplos de Uso (por técnica)

Charter-Based Testing

Exemplo Prático: Em um e-commerce, define-se o charter "Explorar fluxo de pagamento focando em validações". Durante 60 minutos, o testador:

- Testa diferentes métodos de pagamento fora da ordem prevista;
- Provoca erros (cartão inválido, saldo insuficiente);
- Interrompe o processo (fecha navegador, volta páginas).

Resultado: Descobriu que ao trocar de método de pagamento, dados do carrinho eram perdidos.

Teste Ad-hoc

Exemplo Prático: Após deploy de sistema de tarefas, testador explora livremente:

- Cria tarefa com 500+ caracteres no título;
- Deleta tarefa enquanto outro usuário a edita;
- Anexa arquivo de 50MB sem validação.

Resultado: Identificou travamento quando múltiplas ações simultâneas ocorrem sobre a mesma tarefa.

Tour Testing

Exemplo Prático: Em app bancário, aplica-se diferentes tours na funcionalidade de transferência:

- Tour do Iniciante: Descobriu que diferença entre TED e DOC não estava clara;
- Tour de Erros: Encontrou mensagem técnica "Error 500" exposta ao usuário;
- Tour de Dados: Valores muito altos causavam erro de formatação na tela.

Resultado: Cada tour revelou problemas diferentes na mesma funcionalidade.

Teste Baseado em Riscos

Exemplo Prático: Sistema hospitalar com 3 dias para go-live, testador prioriza:

- 60% do tempo: Prescrição de medicamentos (descobriu falta de validação de interações medicamentosas);
- 25% do tempo: Autenticação (enfermeiros acessavam dados financeiros indevidamente);
- 15% do tempo: Demais funcionalidades.

Resultado: Focou 85% do esforço em 20% das funcionalidades críticas, evitando riscos ao paciente.

Teste por Heurísticas

Exemplo Prático: API REST de produtos, aplicando heurísticas CRUD:

- Create: Produto com preço negativo passou sem validação;
- Read: SQL Injection funcionou: id=1 OR 1=1;
- Update: Race condition com 2 usuários causou dados inconsistentes;
- Delete: Deletar produto não verificava pedidos ativos, quebrando sistema.

Resultado: Descobriu 4 vulnerabilidades críticas em 2 horas.

Teste Baseado em Personas

Exemplo Prático: Plataforma EAD, mesma funcionalidade "assistir aula" testada por 4 personas:

- Estudante iniciante (16 anos): Botão play não estava destacado, vídeo começava no volume máximo;
- Professor experiente (50 anos): Sem atalhos de teclado, configurações não salvavam;
- Usuário com deficiência visual: Controles inacessíveis via teclado;
- Hacker ético: Vídeos premium acessíveis manipulando URL.

Resultado: 12 problemas de usabilidade + 3 vulnerabilidades de segurança.

Teste de Aprendizado Progressivo

Exemplo Prático: Sistema legado de folha de pagamento sem documentação:

- Semana 1: Cadastro básico → descobriu que CPF duplicado era aceito;
- Semana 2: Cálculos simples → INSS calculado errado em faixa específica;

- Semana 3: Horas extras → adicional noturno não aplicado em finais de semana;
- Semana 5: Exceções → 3 bugs críticos em cálculo de férias.

Resultado: Documentou 15 regras implícitas e encontrou 8 bugs antigos.

Teste de Ataque (Error Guessing)

Exemplo Prático: Chat em tempo real, testador "ataca" pontos fracos:

- Dois usuários editando mesma mensagem → mensagem corrompida;
- Enviar após token expirar → mensagem perdida sem aviso;
- 10.000 caracteres + 500 emojis → aplicativo travou;
- <script>alert('XSS')</script> → XSS executado no cliente;
- 1.000 mensagens em 10s → sem rate limiting (permitiu spam).

Resultado: 4 vulnerabilidades críticas em 2 horas de teste.

Limitações dos Testes Exploratórios (gerais)

Dependência da Expertise do Testador

- Testador júnior: resultados muito limitados;
- Testador sênior: pode focar apenas em problemas familiares (viés);
- Qualidade varia drasticamente entre indivíduos.

Baixa Reprodutibilidade

- Sem documentação imediata, bugs impossíveis de reproduzir;
- Desenvolvedores precisam de passos claros para corrigir;
- *Mitigação:* Gravar tela, documentar passos na hora.

Cobertura Não Garantida

- Impossível saber se "terminou de testar";
- Áreas podem ser negligenciadas inadvertidamente;
- *Mitigação:* Combinar com testes formais, usar checklists.

Inadequação para Regressão

- Automação é muito mais eficiente para testes repetitivos;

- Exploração manual repetida é cara e propensa a erros.

Estimativas Imprevisas

- Tempo necessário é incerto, dificulta planejamento;
- Gestores não conseguem mensurar progresso facilmente;
- *Mitigação:* Usar time-boxing (sessões de duração fixa).

Limitações dos Testes Exploratórios (por técnica)

Charter-Based Testing

- Charter mal definido vira teste ad-hoc sem foco;
- Pressão do tempo pode fazer pular áreas importantes;
- Testadores diferentes interpretam mesmo charter diferentemente.

Teste Ad-hoc

- Crítico: Sem anotações, bugs impossíveis de reproduzir;
- Testador júnior tem resultados extremamente limitados;
- Áreas críticas podem ser completamente ignoradas.

Tour Testing

- Múltiplos tours consomem muito tempo;
- Tours podem sobrepor-se, testando aspectos redundantes;
- Definir quais tours aplicar exige conhecimento do sistema.

Teste Baseado em Riscos

- Áreas "baixo risco" podem ter bugs graves inesperados;
- Análise de risco é subjetiva (varia entre pessoas);
- Bug pequeno em área "segura" pode causar efeito cascata.

Teste por Heurísticas

- Heurísticas genéricas não cobrem domínio específico;
- Aplicação mecânica pode perder bugs criativos;

- Completar checklist não garante qualidade real.

Teste Baseado em Personas

- Testador não vivencia realidade da persona de verdade;
- Difícil "desligar" conhecimento técnico ao simular iniciante;
- Cada persona requer sessão completa (tempo multiplicado).

Teste de Aprendizado Progressivo

- Curva de aprendizado atrasa descoberta de bugs;
- Quando testador sai, conhecimento pode ser perdido;
- Gestores não entendem por que "demora tanto".

Teste de Ataque (Error Guessing)

- Extremamente dependente: Sem experiência, não sabe "onde atacar";
- Foca apenas em erros já vistos antes (viés de confirmação);
- "Intuição" não é facilmente ensinável.

Quando Usar Cada Técnica ?

Técnica	Melhor Para	Evitar Quando
Charter-Based	Ambientes ágeis, documentação necessária	Exploração totalmente livre
Ad-hoc	Testadores experientes, smoke tests	Testadores júnior
Tour Testing	Sistemas complexos, múltiplas perspectivas	Tempo muito limitado
Baseado em Riscos	Prazos apertados, sistemas críticos	Tempo abundante
Heurísticas	APIs, CRUDs, padrões conhecidos	Domínios muito específicos

Personas	UX, acessibilidade	APIs backend sem interface
Aprendizado Progressivo	Sistemas sem documentação	Prazos muito curtos
Error Guessing	Testadores sênior, segurança	Testadores sem experiência

Recomendações Práticas:

Para Maximizar Efetividade:

1. Combine técnicas: Use charter-based em sprints + teste de riscos antes de releases;
2. Documente imediatamente: Grave tela, anote passos na hora;
3. Pair testing: Júnior + sênior aprendem juntos;
4. Complemente automação: Exploratório descobre, automação regride.

Para Mitigar Limitações:

- Use ferramentas de gravação (OBS, Loom);
- Mantenha checklists de áreas críticas;
- Faça time-boxing rigoroso (sessões 60-90min);
- Compartilhe aprendizados com a equipe.

Referências

BACH, James. **Exploratory testing explained**. *Software Testing & Quality Engineering Magazine*, v. 1, n. 3, 2003.

BACH, James. **Exploratory testing is simultaneous learning, test design, and execution**. Satisfice Inc., 2003. Disponível em: Software Testing for Serious People . Acesso em: 31 dez. 2025.

BLACK, Rex. **Managing the testing process**. 3. ed. Hoboken: Wiley, 2009.

HENDRICKSON, Elisabeth. **Explore it!: reduce risk and increase confidence with exploratory testing**. Dallas: The Pragmatic Bookshelf, 2013.

ISTQB. **Certified Tester Foundation Level – syllabus**. [S.l.]: ISTQB, 2018. Disponível em: International Software Testing Qualifications Board (ISTQB) . Acesso em: 31 dez. 2025.

ITKONEN, Juha; MÄNTYLÄ, Mika V.; LASSENIUS, Casper. **The role of the tester's knowledge in exploratory software testing**. *IEEE Transactions on Software Engineering*, v.

39, n. 5, p. 707–724, 2013.

KANER, Cem. **Exploratory testing**. Florida Institute of Technology, 2008. Disponível em: C em Kaner, J.D., Ph.D. . Acesso em: 31 dez. 2025.

KANER, Cem; BACH, James; PETTICHORD, Bret. **Lessons learned in software testing: a context-driven approach**. New York: Wiley, 2001.

KHATRI, Mohammad Faisal. **Exploratory testing**. *Medium*, 9 ago. 2023. Disponível em: Ex ploratory Testing . Acesso em: 31 dez. 2025.

MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. **The art of software testing**. 3. ed. Hoboken: Wiley, 2011.

SHAH, S. M. A. *et al.* **Towards a hybrid testing process unifying exploratory testing and scripted testing**. *Journal of Software: Evolution and Process*, v. 26, n. 2, p. 220–250, 2014.

WHITTAKER, James A. **Exploratory software testing: tips, tricks, tours, and techniques to guide test design**. Boston: Addison-Wesley Professional, 2009.

YU, Jiujiu *et al.* **Software exploratory testing: present, problem and prospect**. In: INTERNATIONAL ACADEMIC EXCHANGE CONFERENCE ON SCIENCE AND TECHNOLOGY INNOVATION (IAECST), 3., 2021. Anais [...]. [S.l.]: IEEE, 2021. p. 44–47.