

## 10 Steps Toward Better Black Box Testing

---

*Black box testing of financial transactions—that is, testing system input and output without examining the internal workings of the code—may seem mundane and simplistic to some people in your organization. However, when financial transaction testing focuses only on whether or not a product works “correctly” after a change is made, this testing has failed to address one of the most important aspects of black box testing: verifying that the change **makes sense** to your users.*

Another round of code changes means more testing. Your developers have tested their code changes. Your QA/QC team has completed their regression testing. Now it’s time to move beyond focusing on whether or not the changes are *coded* correctly, or that the code behaves as expected, and look at the system from the user’s perspective through black box testing.

Ideally, new features are expected to increase income. However, even the hottest feature may fail to deliver on that economic promise if it is poorly implemented. All too often, users become an organization’s de facto black box testers – and sadly, the new feature fails their tests.

Improve your black box financial transaction testing by incorporating these steps.

**1) Wear your “user hat.”** Good testers assume a variety of roles and, in black box testing, their most important role is that of “typical user.” This is one case in which less is truly more – that is, less knowledge of the code may mean the tester will be able to make a more objective evaluation. In fact, black box testing should include random input, unconventional system navigation, and deliberate entry of unacceptable data to verify the system’s response. Verify that error messages are informative and easy to understand, and that options (such as Cancel buttons) enable the user to back-track or start over easily.

**2) Make certain the solution really fixes the problem** or addresses the need. Most features are added in response to a user need. Does this change really give users what they want? Good black box testing requires an in-depth knowledge of the user and understanding of the user’s needs. Without that affinity for the user, the tester can only ensure that the code does what the developer expects it to do, rather than verify that it actually meets the user’s needs.

**3) Strive for simplicity.** For the typical user, easy-to-use features are easy-to-love features. How complex is the feature? Is it designed for power users only, or is it available to all users? The more accessible the feature is, and the more straightforward its operation, the more likely your users are to incorporate its use into their routine.

**4) Consider the big (or small) picture.** Looks aren't everything, but they do matter. Is the screen design attractive? Does it seem overly busy? Is the new feature hidden, or easy to find? Are fonts too large or small? Is the color palette suitable? Overly busy screens hide new fields and features, making them difficult for users to find.

**5) Go with the (transaction) flow.** Sometimes the rush to market means features are inadvertently implemented outside the normal transaction flow. Is the new feature available when the user needs it, or does it occur too early or too late in the transaction flow? Also, sometimes ATMs collect several input items before sending the input on to the host. For example, does it make sense to have a user enter a PIN, select a transaction, select an account and amount, and *then* be told that the transaction is denied because the PIN failed, or is it more logical to verify the PIN *before* allowing the user to proceed? Adding an extra processing step may mean saving time and reducing frustration for the user.

**6) Document, document, document.** Document your test plan. Document each step you take. Document how and when each issue occurred so the developer can later recreate the scenario. Use text and screen captures to provide developers with as much information as possible.

**7) Don't forget the basics.** Check for spelling and grammatical errors. Verify that the changes follow your organization's rules and conventions for applications. When users encounter "obvious" mistakes, that experience may adversely affect their perception of your organization as a whole.

**8) Report all errors.** Don't make the mistake of identifying "only the important errors." Oftentimes you will be unable to determine which errors are related to significant problems. Better to mention *all* errors and let the developers evaluate them than to explain later why you failed to report an issue that had more severe consequences than you realized.

**9) Focus on facts.** Present your evaluation in the most objective and unemotional language possible. No one wants to think their work is flawed, or that they have made careless mistakes. Identify issues without assigning blame or implying ignorance. Foster cooperation and mutual respect so you and your developers can function efficiently as a team.

**10) Recognize your limitations.** It may sound like heresy, but it is important to remember that no application can be *completely* tested. Every delay in development affects the time remaining for testing, squeezing the test schedule ever tighter. The best testing methodology uses risk analysis to determine where errors can cause the most damage and focuses test efforts there, then supplements with more broad range testing.

And finally, a word of encouragement to all financial transaction testers... Despite the sometimes adversarial relationship between developers and testers, they do share a common goal: produce the best product in the least amount of time. As a tester, you are charged with giving each new feature the final stamp of approval. In essence, by thoroughly evaluating the usability and performance of the feature, you guarantee that everyone

involved successfully reaches their goals. Coding errors can result in loss of income as well as loss of credibility for your organization. You are your organization's insurance against those losses. Take pride in the important role you play in your organization's success.

## About Paragon Application Systems

Paragon Application Systems is a leading global provider of ePayment simulation, configuration and testing software tools to the financial industry. More than 400 financial institutions in over 80 countries use Paragon tools to improve quality and reduce time-to-market. Paragon's broad customer base includes major interchanges, processors, leading software providers, banks and credit unions. Visit Paragon Application Systems at [www.paragonedge.com](http://www.paragonedge.com).

## About the Author

Lynn Tjepkema, ePayment Specialist in Paragon's Professional Services department, is responsible for analyzing customer requests and developing working solutions to address testing needs. Lynn is ISTQB certified and has over 20 years of software development and testing experience in the financial services industry. She has extensive knowledge of financial network message protocols as well as authorization and settlement processing, with special emphasis on Visa, MasterCard and American Express.