testuff

THE 6 THINKING HATS OF SOFTWARE TESTING

TABLE OF CONTENTS

INTRODUCTION

The term “lateral thinking” describes a problem-solving methodology in which users employ reasoning that may not be immediately obvious. One approaches each challenge by stepping ***outside*** the box in order to gain perspectives that might otherwise remain hidden.

Maltese consultant and author, Edward de Bono, first coined the term in the 1960s. And he went on to promote this cognitive approach in his seminal *6 Thinking Hats* – a book designed to help project managers overcome unique challenges in the competitive business climate of 1985.

According to this philosophy, the best way to discover and resolve issues in a given project is to adopt 6 complementary thinking approaches. De Bono ascribed a colored Hat to these lateral thinking styles – each of which should be worn throughout the entire project development cycle.

Those 6 color-coded Hats include:

- **Managerial Blue.** What is the goal? What is the subject? Why are we here?
- **Informational White.** What are the facts? What are our resources?
- **Emotional Red.** How do I feel about this project? What does my gut say?
- **Discerning Black.** Where are the danger areas? What can go wrong?
- **Optimistic Yellow.** Where are the opportunities? What are the benefits?
- **Creative Green.** Can we make this better? Can we incorporate new ideas?

According to de Bono’s research, teams that incorporate these 6 thinking styles:

- Make the fewest mistakes.
- Identify the greatest opportunities.
- Cover the largest terrain.
- Enjoy the most success.

Over the past 3 decades, de Bono's lateral thinking approach has spread beyond traditional business circles to find a ready audience in countless other disciplines – including software testing.

In fact, lateral thinking's impact on software quality assurance has been so powerful that de Bono's revolutionary philosophy has graduated from an enviable competitive advantage to an absolute business necessity. Software testing teams that ***fail*** to don all 6 of his cognitive thinking Hats run the risk of launching inferior products in today's unforgiving IT climate.

That's a bold statement.

How could a ***business*** concept first pioneered in the 1960s hold such sway in the software testing world of the new millennium?

The following chapters of this e-book help to illustrate the enduring power of de Bono's 6 Hats – ***specifically*** as they relate to software development and testing.

Each one of these sections is relevant on its own merits. And many QA teams are comfortable aligning their entire project management approach to individual or discrete pairings of the following Hats. But according to de Bono, long-term success is only guaranteed for those teams that employ all 6 of these cognitive approaches in tandem.

Let's take a look.

CHAPTER 1: THE MANAGERIAL BLUE HAT

Although there is no official order to de Bono's lateral thinking framework, we recommend beginning with the Blue Managerial Hat – a starting point in which teams outline the ultimate goal of the project.

Wearing the Blue Hat allows software testers to ask the most important questions. Namely:

- What is the subject?
- Why are we testing this facet?
- What is our goal?

The Blue Hat is top-level:

- The navigator who charts the direction.
- The conductor who leads the orchestra.
- The thinker who ***thinks*** about thinking.

The Blue Hat wearer decides when tests begin and end – and under what conditions these tests are run (and why).

The Blue is the WHY of it all. And the project's ultimate success largely hinges on how testers choose to ***define*** success at this stage of the process.

Application

In software QA terms, most of the Hats in de Bono's framework have specific testing methodologies attached.

Blue is one of the exceptions to that rule.

When you wear this Hat, you're not as concerned about individual steps or tactics. And you don't let nitty-gritty details bog down the process at this stage. Instead, you remain focused on the overall strategy:

- What are we building?
- What does the end-user expect?
- What is OUR role in delivering that outcome?

Answering these questions will help you decide the best methodologies or tools later on – once you don ***other*** Hats.

Benefits

All of the Hats in de Bono's framework are vitally important. You need 6 modes of thinking to cover your bases. But Blue is the foundation. And you should wear this Hat during all initial planning stages of any project you begin. Doing so ensures that you're only committing resources in a manner (and to a project) that make sense:

- If you get this step ***right***, all subsequent steps align more smoothly. There will, of course, be obstacles and setbacks along the way. These are inevitable. But Blue Hat planning offers a roadmap that allows you to overcome those hurdles. With a destination in mind, you'll know which mountains to climb and what rivers to cross. Blue Hat thinking removes much of the guesswork from the equation.

- If you get this step ***wrong***, your project will join the countless other brilliant ideas that couldn't find a ready market. When roadblocks emerge, you won't have the resources to circumvent them. And you'll waste precious time climbing mountains that don't need climbing or crossing rivers that don't need crossing. Worse still, challenges that actually ***need*** your attention remain untouched.

Key Takeaway

The Blue Hat is the cornerstone of lateral thinking. It may not offer the Eureka moments associated with brilliant problem-solving. But it ***does*** allow team members to crystalize their understanding of goals.

Equally important, Blue Hat thinking makes it easier to identify and remove faulty assumptions that could potentially hinder success. It reduces the likelihood of building something that no one needs or wants.

Do <u>**NOT**</u> begin any project unless someone on the team is wearing Blue. And ideally, this Hat should remain within easy reach ***throughout*** the development and testing cycles. It serves as a useful starting point. But much of the Blue Hat's power comes from its ability to reorient teams should they meander off course later on.

CHAPTER 2: THE INFORMATIONAL WHITE HAT

If the Blue Hat allows you to select a destination, then wearing the White Hat is when you begin taking inventory of the tools and resources required to ***reach*** that destination. Objective and impersonal, White Hat lateral thinking demands that you become a census-taker who determines the facts as they are – and not necessarily as you might ***like*** them to be.

More specifically, White Hat thinking is when you ask:

- What do we know already?
- What do we need to find out?
- How will we obtain these missing data?

There are zero emotional judgments attached to this process. You adopt a strictly clinical approach in determining what pieces of information are verifiable – and what pieces are based on assumptions. If and when gaps emerge, White Hat thinking allows you to identify avenues for retrieving the information you still need.

In short, you want to remove as many assumptions and theories from the table as possible. Other Hats in de Bono's framework offer plenty of opportunities to explore and get creative. But the White Hat is about taking stock and making sure you begin your journey from a place of absolute certainty.

This step is so critical that many projects begin with the White Hat ***instead*** of the Blue. And there may exist times when software testing calls for this reversal. But we believe that asking "why" is usually more important than asking "what."

Let's explore White Hat thinking in action.

Application

Software testing often brings about a lot of emotions – especially as deadlines approach, personalities clash, and budgets shrink. White Hat thinking is neither the time nor place to indulge such distractions. Your main concern is to determine neutral facts like:

- What ***is*** the budget (not what would you like the budget to be)?
- What ***does*** the product currently do (not how can it perform better)?
- Where ***does*** the product currently fail (not why does it fail)?

When you're done with White Hat thinking, you should be able to print out a simple 1 or 2-page report that outlines the current state of development.

"This is where we are."

However, getting to this stage still requires ***some*** testing:

- Focus on regression testing to identify what works and what doesn't. But save exploratory testing for later on when you're ready to figure out the ***why*** of whatever bugs you uncover.

- Use automation to cycle through and pinpoint known issues. But save manual testing for later on when your quest to uncover new opportunities requires direct human intervention.

- Use pre-determined scripts and schedules to verify what is known. First iteration tests should last as long as second and third run-throughs. You're not here to find ***new*** information. Your goal is to make sure that components stay working – or stay non-working.

Benefits

The best way to understand why White Hat thinking matters is to consider instances when this critical step is absent.

Thanks to ***Blue Hat*** thinking, your team understands the basic goals of the project. You all share a common roadmap. But because you haven't gone through the ***White Hat*** fact-collection stage:

- Half the team thinks that the budget is A – the other believes it to be B.

- Half the team thinks that component X is working – the other half realizes that it still needs fixing.

- Half the team is working to improve feature Y – the other half understands that this feature isn't part of the beta release.

When you have competing assumptions within the same team, then delays and frustrations are inevitable. White Hat thinking removes speculation from the testing process, so that everyone is on the exact same page.

Key Takeaway

Most software testers feel right at home wearing the White Hat. Ours is a data-driven profession. And whether consciously or not, most of us automatically take stock of what we know – before jumping into any big project.

However, avoid the temptation of tinkering at this stage. New ideas and insights ***will*** begin forming in the back of your head (i.e. "If A is true, then B may be the cause"). But there will be plenty of time to test those assumptions in later stages.

For now, just the facts – and nothing but the facts.

It's okay if White Hat thinking doesn't come naturally. After 1 or 2 ***failed*** projects, it will. You'll intuitively grasp the benefits of fact-collection. And you'll begin to see the White Hat as a testing methodology in and of itself. In many ways, it's the most valuable tool in our arsenal.

But as you'll soon see, objective analysis can only take a project so far (otherwise, machines would do all of our software testing for us).

The next chapter illustrates the importance of the ***human*** touch – imperfect and subjective though it may be.

CHAPTER 3: THE EMOTIONAL RED HAT

Many regard software testing as a strictly clinical discipline, devoid of emotional attachment. And yet, who among us hasn't become frustrated by a particularly stubborn fix – or felt the elation of finally finishing a difficult project?

However, these powerful experiences are usually unpredictable. We become overwhelmed and caught off guard by our emotions.

But according to Edward de Bono, there are times when we must ***deliberately*** tap into the intuitive or passionate portions of our being. This inward exploration of gut feelings allows us to understand potential opportunities and challenges that our rational selves might normally overlook.

This is why all teams should, at one point, adopt Red Hat cognitive thinking – a strictly emotional process in which you avoid logically justifying why you feel the way you do. Coming up with explanations at this stage doesn't help move the process forward.

There are ***other*** Hats for that purpose.

For some software testers, letting go can be difficult. We're analytical by nature and desperately want to understand the inner-workings of whatever problems we confront. Speaking in the language of "feelings" may seem like a foreign concept.

But there are important advantages of tapping into these emotions.

Let's take a look.

Application

Red Hat thinking starts and ends with one simple premise.

How does the project make you feel?

When you ask this question, you open your mind to potential pitfalls and hidden opportunities. These are not always things that you'll uncover with deliberate scanning. Rather, Red Hat thinking allows you to call upon your gut instincts. There may be a nagging feeling or itch that you can't exactly articulate – but you ***know*** it's there.

That itch could be something as simple as a personality conflict with one of your team members. Or perhaps you feel that the developers keep messing up the GUI. Or maybe you suspect that the product will never be ready for prime time.

No proof is required to justify these feelings, but you'll have that nagging sensation nonetheless.

But how is any of this beneficial?

Benefits

Red Hat thinking encourages you to conduct tests that you might not otherwise try. You start the process according to what you ***feel*** is wrong. You're letting intuition be your guide – rather than relying on rigid, predefined steps.

As a methodology, exploratory testing is extremely conducive to this Red Hat philosophy. The process is very intuition-driven. But if you ever get stuck, a good rule of thumb is to think like the end-user. See the product through his or her eyes:

- Focus on aesthetics, usability, and benefits.
- Will the end-user enjoy using this product?
- Are there any aspects that could result in unforeseen frustration?

Don't underestimate the importance of this exercise.

History is full of products that ultimately failed – even though they delivered exactly as promised. Having a fully functional prototype isn't enough. The end-user can still develop very negative feelings about using your product. And given how unforgiving the market is, you may never have an opportunity to make a second 1st impression.

Key Takeaway

Red Hat thinking isn't a standalone solution. It complements the other 5 Hats in de Bono's framework. And you don't necessarily need to come up with a workaround at this stage. In fact, de Bono himself recommends devoting only limited resources to Red Hat analysis.

But that doesn't mean you should gloss over this step. Machines may be faster and smarter in many respects. But they can't approach us when it comes to emotional scanning. This is a uniquely human trait. And when employed effectively, the Red Hat can spark some of the greatest lateral thinking insights.

When you're ready to finally articulate and fix whatever is wrong, it's time to put on the Black Hat.

CHAPTER 4: THE DISCERNING BLACK HAT

When it comes to search engine optimization (SEO) and network security, the term "black hat" has very negative – often ***illegal*** – connotations. But in project management circles, wearing the Black Hat can be a good thing.

Not just good.

Done correctly, Black Hat thinking can play an invaluable role in long-term success. And it is while wearing this Hat that many software testers feel most at home. That's because this cognitive approach requires a mix of critical judgment and logical analysis.

In effect, we must employ negative thinking to produce positive results.

It's not that software testers are pessimistic by nature. But we ***are*** trained to figure out why things don't or won't work. Give us a situation, and we'll find as many holes as possible. This is, after all, what we are ***paid*** to do. As gatekeepers of the realm, our job is to apply objective logic and find leaks. This cautious conservatism ensures that the final release of every product is user-friendly and free of bugs.

But it's not simply negative logic that we're using.

All of our findings and assumptions need to be backed up with facts. Whereas Red Hat cognitive thinking allows you to indulge potential pitfalls on an ***emotional*** level, Black Hat analysis requires verifiable data and testable justifications.

Application

There are many different ways to apply Black Hat analysis. By definition, almost ***any*** methodology can be used to scan for and identify aspects of the product that need improving.

However, below are some general guidelines for Black Hat software testing:

1. During the Actual Testing Process

Use Black Hat thinking to discover how the testing process may be flawed. Potential approaches include:

- Finding production problems and defects.
- Identifying parts of the system or application that haven't been tested.
- Benchmarking version releases, postponed tests, and missed deadlines.

- Verifying whether "fixed" bugs require reopening (if they weren't properly debugged).
- Manually checking bugs that were missed during automation.
- Ensuring that time, money, and other resources are allocated responsibly.

Using data collected from the above steps, your team can improve the overall testing process. Black Hat thinking helps you identify problems and inconsistencies in your quality assurance workflow.

2. Different Types of Software Testing

Teams should employ Black Hat cognitive thinking to prepare future tests and select the most appropriate methodologies:

- Do you have sufficient information to begin testing?
- Do you know precisely what is different about each version of the product?
- Do you have the personnel and resources to run the required tests?
- Have there been any changes (internally or externally) about the product that your team isn't aware of?

3. Replacing Assumptions with Verifiable Facts

Teams should use Black Hat software testing to critically examine any and all assumptions – especially ***optimistic*** ones. This step is arguably the most important since failure to spot "holes" is what ultimately makes or breaks a product.

- Is the product (or system) user-friendly and intuitive?
- Does it include the most important features – especially those specifically requested by users?
- Does this product have a ready market? And if not – why not?

Many products launch with absolutely zero bugs. And yet they ***still*** fail in the marketplace. This is because software testing isn't simply about finding errors. It's about delivering solutions that customers desperately want and need to use.

Benefits

Edward de Bono cautions against spending too much time on emotional Red Hat thinking. But when it comes to the Black Hat, be prepared to invest as many resources as necessary.

We couldn’t agree more – and here’s why.

Human nature is funny. Our brains are wired to scan for whatever we ***tell*** them to scan for. If we look for 3 potential problems, we’ll always find exactly 3 (maybe more). It we look for 10, we’ll find 10.

In other words, it pays to set unrealistically high targets to ensure that you completely cover your bases. Instruct your team to look for 25, 50, or even 100 reasons why a project won’t work.

Key Takeaway

Black Hat thinking asks you to assume that ***everything*** is assumed. Take nothing for granted. And work backwards until you’re on 100% solid ground.

But note that the Black Hat isn't about ***solving*** problems. The goal is simply to identify their existence. At this stage, don't get mired in debates or arguments. The Black Hat approach allows you to objectively outline all of the negative elements throughout the product cycle.

In the next chapters, we’ll look at how de Bono recommends you resolve all of these problems and inconsistencies.

CHAPTER 5: THE OPTIMISTIC YELLOW HAT

By definition, software quality assurance is a highly analytical discipline. As you test and debug different products, you rely more on the left hemisphere of your brain – the side that handles logic.

If there were a color associated with this analytical approach, it would probably be a neutral gray – devoid of any emotional or subjective undertones.

And yet, software testing can be a highly optimistic and positive field, with successful testers drawing on the ***right*** hemisphere to find harmony within their builds. Problems are simply challenges yet overcome. And wearing the Yellow Hat allows you to turn those challenges into victories.

More specifically, wearing this Hat demands that you ask:

- What benefits exist that may have been overlooked?
- How can you improve whatever doesn't work?
- Can you isolate the value within the product?

Note that this is ***mostly*** exploratory.

Yellow Hat thinking doesn't require that you evaluate potential proposals or examine competing solutions. You're simply collecting as many good ideas as you can. It's okay to ***casually*** test feasibility if time permits. But it's best to treat this stage the way you would a group brainstorming session.

In other words, there are no "bad" ideas. The focus is on positive and constructive feedback that is all future-oriented.

Terrific. But how do you put this into practice?

Application

When it comes to execution, Yellow Hat thinking most closely resembles the Red Hat. There is no formulaic approach. Every team, every product, and every launch will follow a slightly different path. And collectively, you will use whatever experience and information you have at your disposal in order to generate ideas.

But done correctly, those ideas will beget new ones – again and again.

What does “done correctly” mean in this context?

It helps to use a non-software example to illustrate this point.

Imagine an artist painting a still-life portrait. He might move about the room in order to view the subject from different angles. By altering perspectives, the artist discovers new ways of capturing the underlying value of his subject – ways that ***would*** have remained hidden had he stayed stationary.

Applying this to software testing, your team might:

- Use new or counterintuitive testing methodologies that you wouldn't normally have considered.
- Find ways to improve older testing processes by shortening them, streamlining them, or doing them in reverse.
- Isolate ways to reduce testing times or improve overall productivity.
- Replace or update certain testing tools in order to take advantage of newer features.
- Bring in team members from outside of the testing department. This is how you transform "inside-the-box" into “outside-the-box” – a critical component of de Bono’s lateral thinking framework.

Benefits

The most obvious advantage of Yellow Hat thinking is that it allows you to resolve issues discovered during the Black Hat process:

- Before, you found tons of problems and reasons why the project ***can't*** work.
- Now, you're isolating potential solutions and reasons why the project ***will*** work.

Again, the goal isn't to fully develop these solutions. Rather, you are exploring how each suggestion could improve the testing process (and by extension – the product) in some demonstrable way. And you are also isolating what barriers need to be removed in order to make that positive future a reality.

Key Takeaway

Constructive optimism is the name of the game. And as such, there is no set time limit for the Yellow Hat. Wear it as long as necessary. And be prepared to revisit this Hat if and when new obstacles emerge (and they most certainly ***will*** emerge).

You should also keep in mind that new solutions often create new problems. So you'll need to sometimes take a step back, put on the Black and Red Hats again, and then dive ***back*** into Yellow Hat thinking.

Do this enough times, and your product will be on fairly secure footing. But you’re not out of the woods quite yet. There is still one final step in de Bono's framework.

And it is literally the most lateral.

CHAPTER 6: THE CREATIVE GREEN HAT

Green Hat cognitive thinking focuses on growth and fertility. Team members should concentrate on:

- New ideas.
- Beneficial changes.
- Unexplored alternatives.

Given all of the problems discovered (or undiscovered) to date, what creative steps can one use to resolve these remaining issues?

In other words, what is possible – and how precisely do you get there?

An outsider might have a hard time reconciling how creativity meshes with software testing. After all, ours is a highly analytical field that requires critical thinking and logic – i.e. what many consider “classical problem-solving.”

In effect, software testing is a science.

But it’s also an art form – and it is largely Green Hat thinking that makes it so:

- The very act of ***finding*** problems requires an outside-the-box perspective that defies common sense. One must approach each task with a child's mind that is open to new possibilities – both obvious and counterintuitive.

- The act of ***resolving*** problems requires even greater flexibility since, by its very nature, software testing is all about exploring new ground. We often find ourselves answering questions that have never been asked.

This explains why we’re such strong advocates of continuous training. The goal of courses, conferences, and self-study isn't to make you ***more*** analytical. Rather, the goal is to expose your mind to new ways of doing things:

- Mastering a ***single*** methodology will make you a formidable expert to be sure.

- Dabbling in ***countless*** methodologies helps you create connections beyond reach of the most experienced veterans.

And this applies to more than just methodologies.

Green Hat thinking demands that you constantly try new automation approaches, new tools, new tests, and even new ways of outsourcing. And not just as a theoretical exercise to ***hone*** your skills – but actually *in situ* with each project your team tackles.

Let's explore how this works.

Application

At the very least, you should begin running tests not already performed – even if those tests aren't normally used to discover the types of bugs you'd expect to find.

However, this is the bare minimum.

True Green Hat thinking goes much further than adopting unusual methodologies:

- Try reducing or expanding the scope of all tests. And adjust how you interpret the results.
- Reassign testers randomly throughout the development cycle so you benefit from a fresh pair of eyes at each stage.
- Involve software developers (yes – ***those*** guys) at random stages throughout the testing cycle.
- Run the same tests as before – but use a completely different set of tools.
- Use the ***same*** test and tools, but now run everything in reverse (i.e. last tests first – and first tests last).

Benefits

The advantages of Green Hat testing are tremendous. With the ***current*** project, you can find and debug all kinds of errors that might otherwise remain hidden.

But this approach also has the potential to reduce development times – not only for today's project, but also for tomorrow's.

You might accidentally invent a new methodology that can be applied to future builds. Or you could uncover a faster way to suss out bugs or a cheaper method for fixing errors.

Key Takeaway

Green cognitive thinking isn't a cure-all for all of your testing woes – and you shouldn't expect to find hidden nuggets all of the time.

What you ***will*** find, however, are ideas and approaches that you never would've tried otherwise. Wearing the Green Hat divorces you from "groupthink" and pushes you out of the traditional comfort zone.

And this is ultimately the goal of lateral thinking – to visit unfamiliar territory that classical analysis would never (and could never) explore.

You won't strike gold each time. But every now and then, you ***will*** discover something new. And with that insight, you can begin pulling the thread and developing innovative solutions to problems you never knew existed – or making improvements you never thought could be made.

Note, however, that in isolation, Green Hat testing isn't very useful. There is little value in pushing improvements if the entire platform exists on shaky ground. But when successfully combined with the ***other*** 5 Hats in de Bono's framework, this cognitive thinking style can make the competition turn Green with envy.

CONCLUSION

When de Bono first introduced the concept of lateral thinking, he offered it as a contrast to traditional problem-solving. Instead of following rigidly defined logical steps, one should think ***outside*** the box. By relying on intuition and creativity, teams could stumble upon solutions that might otherwise remain hidden.

Lateral thinking, however, isn't a switch you can turn on or off at will. Eureka moments typically come out of nowhere – they can't be forced into existence.

But de Bono ***does*** offer a formula that makes it easier to coax out brilliant insights. By adopting his 6 Hats, you can consciously structure the development cycle to make rigid thinking less tenable – and lateral thinking more achievable.

Note, however, that de Bono's *6 Thinking Hats* is a framework and not a step-by-step guide. The process is iterative, requiring that your team wears all 6 Hats periodically (and repeatedly) from Day 1 until the final release.

But for the vast majority of testing situations you may encounter, there does seem to be a logical order regarding ***when*** to wear each Hat. Although this sequence may not apply in every scenario, it's a useful guideline nonetheless.

Step 1: It is generally recommended that you decide on the best tools and methodologies from the very beginning (even ***before*** you start testing). While wearing the Blue Hat, discuss the entire developmental process from beginning to end – including a careful analysis of resources (e.g. testers, equipment, environments, and skills).

Step 2: While wearing the White Hat, assess whether those resources are enough to move forward. Note that you still haven't started testing yet. The team is merely finalizing important metrics like the scope, budget, and timeline of the project. Although having a consensus is ideal, this won't always happen. At the very least, however, everyone should understand the goals and guidelines – even if they don't necessarily "agree" with each detail.

Step 3: Finally you can begin testing, with the ultimate goal of finding and fixing defects. A typical approach might be to segment your team into 2 groups, each of which is in charge of a particular Hat. For example:

- The Black Hat group adheres to the preselected methodologies isolated in earlier steps. This group validates each fix, handles all regressions, and essentially does "normal" testing.

- The Red Hat group focuses on exploratory testing, opening up the development process to gut feelings and intuition. They are temporarily allowed to indulge their emotions if doing so better enables them to understand each build from the user's perspective.

Step 4: The project is finally error-free – at least with the current version. You can now begin exploring ways to improve ***future*** tests. When wearing the Yellow Hat, your focus is on the most recently completed project. But all improvements should keep pipelined projects in mind as well. That way, you can continue to evolve and spot new opportunities later on.

Low-hanging fruit include:

- Finding errors in less time (and with fewer resources).
- Streamlining and refining methodologies.
- Updating (or replacing) older testing tools.
- Rotating and training team members.

Step 5: Last (but not least) is the Green Hat. The team should set aside time periodically to collect feedback from as many different stakeholders as possible. These brainstorming sessions can ultimately power your wider R&D. Not every idea on the table will be feasible. But wearing the Green Hat allows your team to uncover winning ideas and allocate the requisite resources to bring those opportunities to fruition.

Follow these loosely defined guidelines, and you'll be amazed at what you're able to accomplish.

Arguably more important, you'll amaze your end-users as well.

testuff

www.testuff.com