

# Six Testing Hats

by Julian Harty, Google (EuroSTAR 2006)

The “six-thinking hats” concepts from Edward de Bono are something I’ve started to use this year that seem to help me, and colleagues, to improve our testing in a number of ways. In the process of adapting the concepts I have built on other ideas and concepts from within and outside the software testing community; and the idea of six testing hats helps me to remember some of these additional concepts. Read on to learn more about the background and concepts, and to answer questions such as:

- Why 6?
- So what is a software testing hat?
- But will it help me with my testing?

The hats provide a metaphor for six different and distinct viewpoints. Each hat has a colour associated with it, rather than a complex style or name. By having simple, plain colours the hats are easy to recognize and understand in many countries of the world.

The six hats are:

Hat	Rationale	Comments
Blue	Control and organization	Provides focus. Helps to monitor progress.
White	Factual viewpoint	Concentrates on obtaining and finding facts and figures
Yellow	Positive and speculative	Constructive thinking, encouraging
Red	Emotions and feelings	Intuitions and hunches
Black	Cautious and careful	Concentrate on what can go wrong, protective
Green	Creative thinking	Providing fresh alternatives, without criticism

The hats may be physical, or imaginary. Regardless, they help us to quickly set or change the direction of our thoughts. Their main purpose is to help us direct our thoughts and ideas in a particular direction. Hats can be swapped quickly, sometimes once a minute, when working alone, or in small, cohesive teams.

## Improving our working relationships

Software development practices may be viewed as adversarial for instance when software is ‘thrown over the wall’, or when groups blame each other for problems and delays. For instance: ‘If you’d given us better specs, we wouldn’t have had to spend an extra 3 weeks and 50,000 Euros trying to find out what the users needed...’

By adopting the principles of the six-thinking-hats we are able to disentangle facts from emotions (white and red hats), to present careful and cautious views (black hat) balanced by a mix of positive thinking (yellow hat) and creative suggestions to improve things (green hat). And with our blue hats we can consider the ‘bigger picture’ of how our work fits with the project and company objectives. If the other people we are working with don’t know about the hats we can choose to use more neutral terms to describe our ideas and thoughts in each area.

As others come to see the benefits of our work we have the opportunity to explain how we managed to improve our outlook and our work by using the hats. Hopefully others will start to use the hats as well, which will help to further improve the communications within teams.

## Using the six-thinking-hats in reviews

A key skill is the ability to review artefacts related to software. Everybody who’s involved in software has to review material of one sort or another, ranging from rough requirements documents to thousands of lines of source code, to the test cases, etc. Sadly, we are often less effective than we’d like, and our feedback may be perceived as unhelpful at best. However by using the six-thinking-hats we have an opportunity to improve the effectiveness and quality of our reviews.

Each hat allows us to view the material being reviewed from six distinct directions. And as we can categorize our comments and feedback under the various hats we can reduce the perceived personal ‘attack’ on whoever’s responsible for the material.

### Using the six-thinking-hats to design test cases

Generally our testing fits within a larger context, that of shipping working, desirable software cost-effectively and on time. One well accepted practice is to incorporate risk in order to decide what to test, how to test, and how much to test. The practice includes risk analysis and test design, and is known as “Risk-based testing”.

Risk-based testing is a key driver in deciding what to test, how to design the test cases, and to decide how much testing will be sufficient to balance the risks with the rewards. However, our testing is only as good as our understanding of the risks. In some projects the risks aren’t consciously considered or addressed. In other projects risks are formally captured and analysed very early in the software lifecycle.

One way to improve our understanding of the risks, and then to design appropriate tests is to use the six thinking hats throughout the process. Each hat is used to help improve the effectiveness and appropriateness of the resulting tests. The following table provides examples of questions we created by viewing the problem from the perspective of each hat.

**With the hats we can ask questions such as**

Questions	Hat
• What data do I need to design this test case?	White
• What are the sources of information I need?	White
• What results do I need to collect?	White
• What are the business objectives and requirements?	White
• What do I need to prepare for the tests e.g. in terms of configuring the system?	White
• What advantages do we obtain from designing (or skipping) this test case?	Yellow
• What outcome would we like to achieve?	Yellow
• How can we improve our working relationships with the rest of the project team by doing this testing? (how can we alleviate their pain or worries)	Yellow
• How do I <i>feel</i> about the software being tested?	Red
• What sort of things would annoy me if it didn't work properly?	Red
• What would annoy the users or the customer if it didn't work properly?	Red
• What sort of problems could go wrong with the test? What sort of things might we get wrong, or misinterpret?	Black
• How can we design our test cases to reduce the chances of the tests being inappropriate or problematic?	Black
• What business or economic (money) impact would there be if the software didn't work properly?	Black
• Where are the <i>dangerous areas</i> ? Where do we need to be particularly careful?	Black
• Are there novel ways we can test the software (e.g. using exploratory testing techniques to complement scripted testing)?	Green
• Are there alternatives to the way we currently do our testing that might improve the results, timescales, costs, etc. of our testing?	Green
• Who do we need to work with/involve to get the testing done?	Blue
• What are we trying to achieve with our tests?	Blue
• When will we know when we're done?	Blue
• Who do we need to keep informed about our progress?	Blue
• How should we report our progress?	Blue

### **Using the six-thinking-hats to assess the product**

The method can be used to assess the software being developed, e.g. to identify potential problems, missing functionality, etc. The following brief case study provides an example where the project team reviewed some new software prior to accepting it from their supplier.

### **A case study on using the six thinking hats for UAT**

A small software testing consultancy commissioned a new version of their web site, which was being developed by another company. The project team were responsible for user acceptance testing (UAT). They used De Bono's six thinking hats as part of the UAT process. The initial review, including an introduction to the 'hats' took an hour.

Each member of the team used a template similar to the one that follows.

As they reviewed the current version of the web site, they were encouraged to make notes under the various hats. At the end of the meeting the notes were collated and analyzed. The notes covered various key aspects of the site, including:

- Core functionality: e.g. how to update the content
- Usability: including page size, aesthetics, look and feel, etc.
- Privacy and Security: e.g. who can access information, how well is the client data protected
- Performance: although only in general terms
- Process testing: including the interaction between people and the functionality offered by the web site – who does what, and when...

As a result of the feedback, the launch of the new site was delayed until various changes were made. The entire team also felt much more involved in the success of the project and were (more) committed to helping improve the new site.

### **Using hats to discuss release meetings**

Another consultancy provided examples of how they used the six-thinking-hats to discuss release decisions – should this software go live, or not?

The hats were applied on key findings that were the subject of discussion, for example – defects not repaired yet – and helped to provide fresh insight into whether the software was likely to meet the release criteria.

### **More hats for software testing**

Debates have raged within the software testing community about what is and is not software testing and how to perform software testing effectively. Some people claim we cannot and should not test without requirements, others claim that scripted testing is futile and should be replaced by exploratory testing, for example. One important contributor to the various approaches to software testing is Bret Pettichord, who describes 'schools of testing'. In his original work he identified 4 schools:

- The analytical school: where we take an analytical approach to testing
- The quality school: where testers are the 'quality police' who stop the developers from shipping 'bad code'
- The factory school: which is process-driven to reduce the costs and risks
- The context-driven school: where the testing is based on the current context and circumstances

Subsequent work suggests a fifth school, using test-driven development practices. Test-driven development encourages unit tests to be created before the code is written. The unit tests are then run to demonstrate the software works correctly. A number of variations exist on the concept of putting the tests first.

Sticking to a single school is risky and may artificially limit the effectiveness and productiveness of the testing. Various testing practitioners agree a mix of schools will help to improve the quality of our work.

Exploratory testing combines test case design with test execution, where the next test is often based on the experience of the immediately preceding testing. According to proponents it offers tremendous opportunities to find new bugs, by not being limited to, or constrained by, scripted tests.

And risk-based testing, as mentioned earlier, is becoming more and more popular as a way to decide where, how and how much to test.

Finally in this section, benefits-driven testing tests the software to demonstrate whether specific benefits have been achieved. For example one of the key benefits for the widget site is to implement an online payment module which is expected to deliver significant benefits to the business, by allowing us to easily process orders from new customers who might not want to create a business account with us. Therefore we might want to test the new payments module as a higher priority as it is expected to deliver a key and immediate benefit to the business.

### Mapping testing ideas to 'testing-hats'

We can map these various ideas to create 'testing hats' that combine the concepts proposed by Edward De Bono with some of these ideas from the software testing community. The goal is to help us to improve the ways we test software, by giving us at least six different ways to approach our testing. The overall outcome provides a powerful way to improve our testing!

Testing idea	Hat	Comments
Benefit-driven testing	Yellow	Looking to show the benefits of the software being tested, e.g. showing what works, rather than what is broken.
Risk-based testing	Yellow (and black)	Only test to assess or mitigate risks related to the project.
Factory school	Blue	Testing is only part of a bigger picture and needs: <ul style="list-style-type: none"><li>• To match the business drivers,</li><li>• Integrate with other tasks,</li><li>• To fulfil the project and other objectives.</li></ul>
Quality school	Black	Process-oriented, leave little to chance: for instance we may want to implement regression testing to reduce the chances of old bugs reappearing.
Analytical school	White	Also includes white-box testing techniques, and focuses on applying proven techniques. These may measuring the effectiveness of techniques and even the testing e.g. Dot Graham's Defect Detection Percentage (DDP) measure.
Context-driven school	Green	The tests are based on the context, which tends to be an iterative, evolving, creative process.
Test-first / test-driven development	Red	Commit to testing early, before the main software is written, passionate about testing. Some practitioners are called 'test-infected'.
Exploratory testing	Green, Red	Creative, seeking alternatives and fresh ideas for finding bugs.

From the previous table, which maps the various testing ideas to the colours of the hats used by De Bono, we can now reassess our software testing, and hopefully find additional ways to look for bugs that may have hidden from us in the past.

You can even use the six-thinking-hats to explore the application of each of the approaches mentioned in this section, for instance – what are the advantages, and risks with using test-driven development? By applying each of the hats in turn you may decide the effort in implementing one or more of the schools is worthwhile.

### Over to you

Don't be constrained by the mapping presented here, you may want to adapt the model to suit your needs and interpretation. What's more important than picking a particular colour (or colours) is for you to gain fresh insights and perspectives on how to test effectively and efficiently.

Like any skill, applying the six testing hats is something that improves with practice. Thankfully the learning curve is gentle and short, and you should be able to make tangible improvements within a few hours.