

[Presentation Notes](#)  
[Paper](#)  
[Bio](#)  
[Return to Main Menu](#)

**PRESENTATION**

---

**W8**

---

Wednesday, November 3, 1999  
2:15 PM

---

---

# USING THE ICED T MODEL TO TEST SUBJECTIVE SOFTWARE QUALITIES

---

---

**Andy Roth**

Rational Software

INTERNATIONAL CONFERENCE ON  
**SOFTWARE TESTING, ANALYSIS & REVIEW**  
NOVEMBER 1-5, 1999  
SAN JOSE, CA



## Using The "ICED T" Model

---

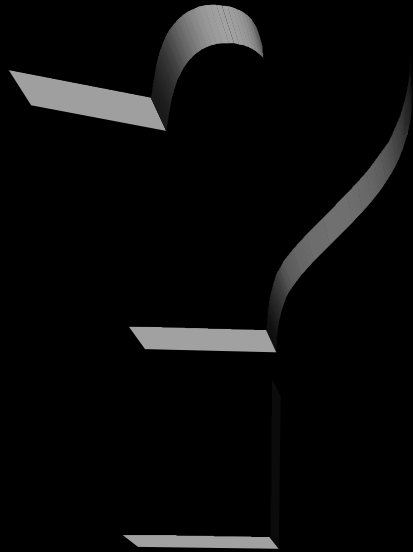
For Testing Subjective Software Qualities

Andy Roth Rational Software

[aroth@rational.com](mailto:aroth@rational.com)

# Presentation Agenda

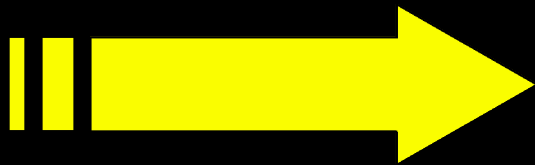
- ◆ **Introduction**
- ◆ **ICED T Model**
  - Why?
  - Model Definition
- ◆ **Using The Model**
  - Testing With The Model
  - Putting A Number To A Feeling
  - Taking the ICED T Plunge
- ◆ **Final Thoughts**



## What Are “Subjective Qualities”?



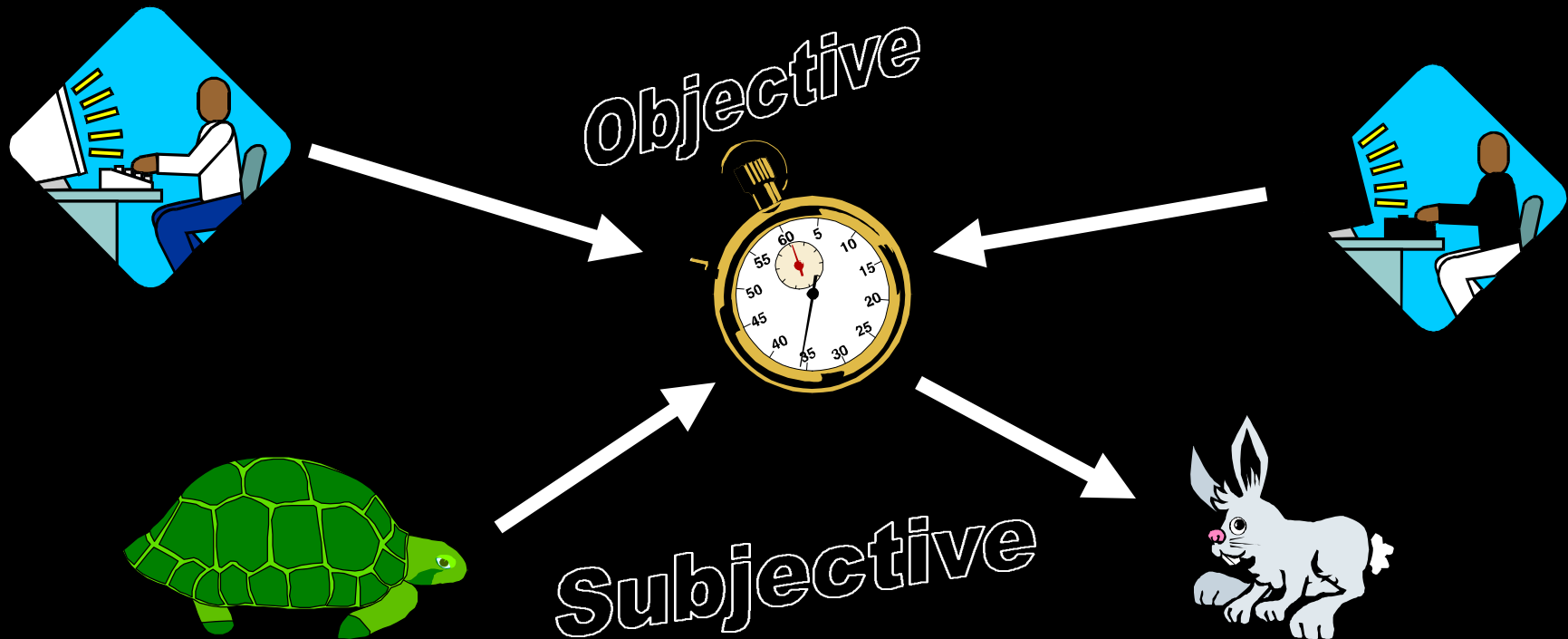
- ◆ **Objective: Absolute**
  - Tend To Quantifiable
    - Yes/No, 2/3/4, On/Off
- ◆ **Subjective: Based on one’s feelings, subject to interpretation**
  - Softer, More Qualitative
    - Good/Bad, Fast/Slow, Easy/Hard
- ◆ **Subjective Qualities are influenced by Objective Measures**



Subjective Qualities are those that are based on the user’s experience of the product and may be influenced by objective qualities.

# How Do They Apply To Software?

- ◆ The User's Overall Experience Of The Application Will Determine Their Evaluation Of Its Quality
  - ◆ This goes beyond just the objective aspects of the software
- ◆ Two People Can Subjectively Evaluate The Same Application Differently Even Though It Is Objectively The Same



# Why Are They Important For Testing?

- ◆ Customers will evaluate the software's quality subjectively
- ◆ By their nature, subjective qualities do not lend themselves to be charted or graphed.
  - This causes them not to be reported
  - If they are not reported, then they probably won't be tested for

# ICED T Model

Consistent  
Thoughtful  
Thoughtful  
Thoughtful

# Why?

- ◆ “Is this software good?” is too broad a question
  - Breaking the subjective measures into categories makes them manageable
- ◆ Better focus for Testers
  - Allows tester to consider all areas of quality
- ◆ Better feedback for Developers
  - Highlighting specific areas of concern allows them to focus their attention on improving them
- ◆ Better feedback for Managers
  - Let them know more specifically where the product stands

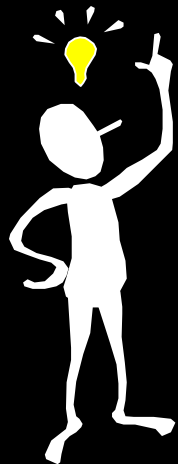
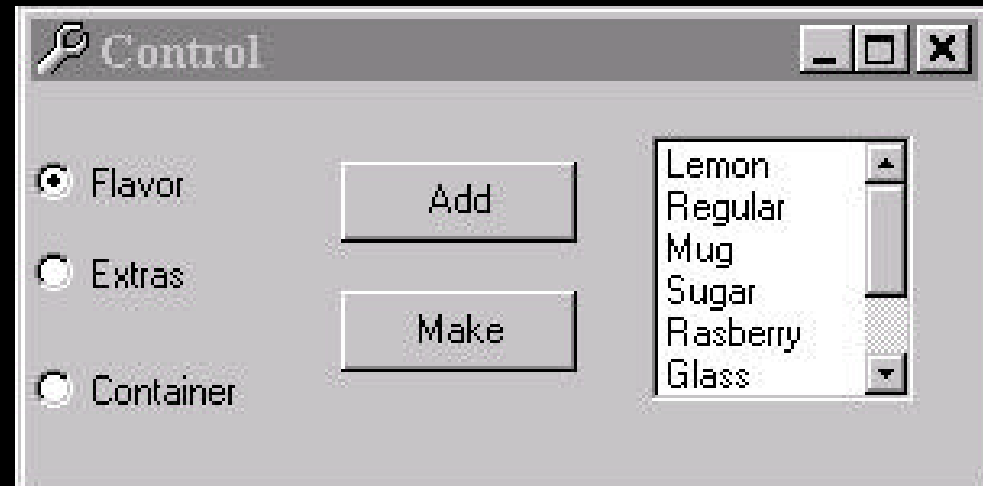
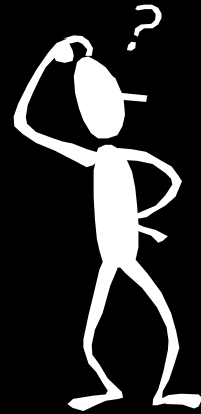


# Intuitive

## Does the use of this product make sense?

- Given actions produce logical results
- Product functions as one would expect
- Software's design implies its use

# Intuitive



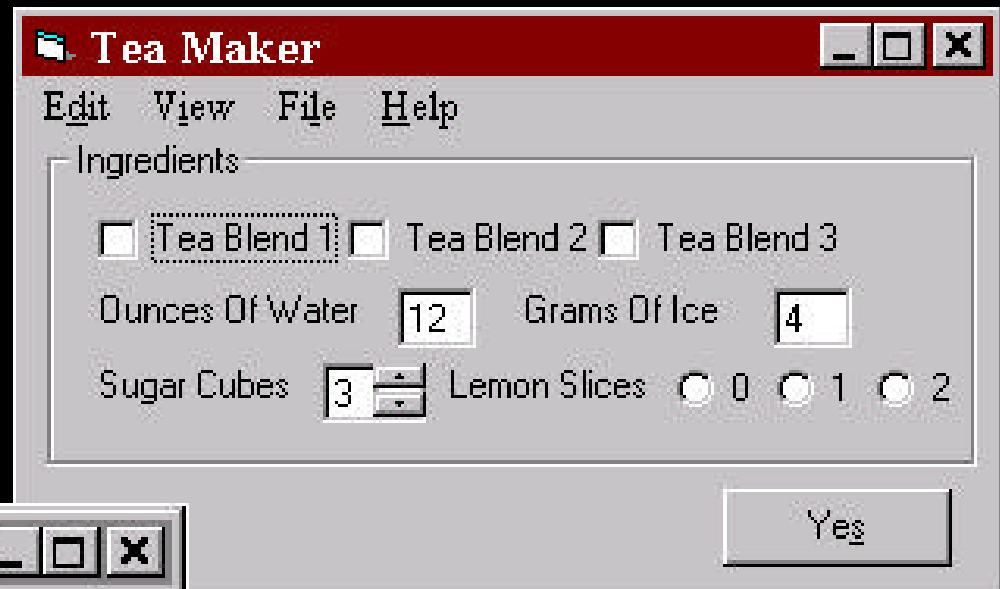
# Consistent

Does the product operate in a uniform manner?

- ◆ **Three levels of consistency**

- Application is consistent with itself through time
- Application is consistent with itself throughout the UI
- Application is consistent with other applications

# Consistent



# Efficient

Is the product quick and swift to use?

- ◆ **Efficiency of the interface**

- Are shortcuts and hot-keys utilized effectively?
- Can any redundancies be eliminated?
- Is the application designed so that it requires a minimal amount of action and time to navigate?

- ◆ **Efficiency of the code**

- How quickly application responds to user's input
- Need to evaluate in a similar context to what the end-user will be using

# Efficient

**Order Tea Blends**

Shipping Address

First Name

Last Name

Address 1

Address 2

City

State

Zip

Billing Address

First Name

Last Name

Address 1

Address 2

City

State

Zip

OK

**Order Tea Blends**

Use An Existing Account    UserName     Password     OK

Remember Password

Enter Information

Billing Address

First Name     Last Name

Address 1

Address 2

Zip

City     State

Ship To This Address

Shipping Address

First Name     Last Name

Address 1

Address 2

Zip

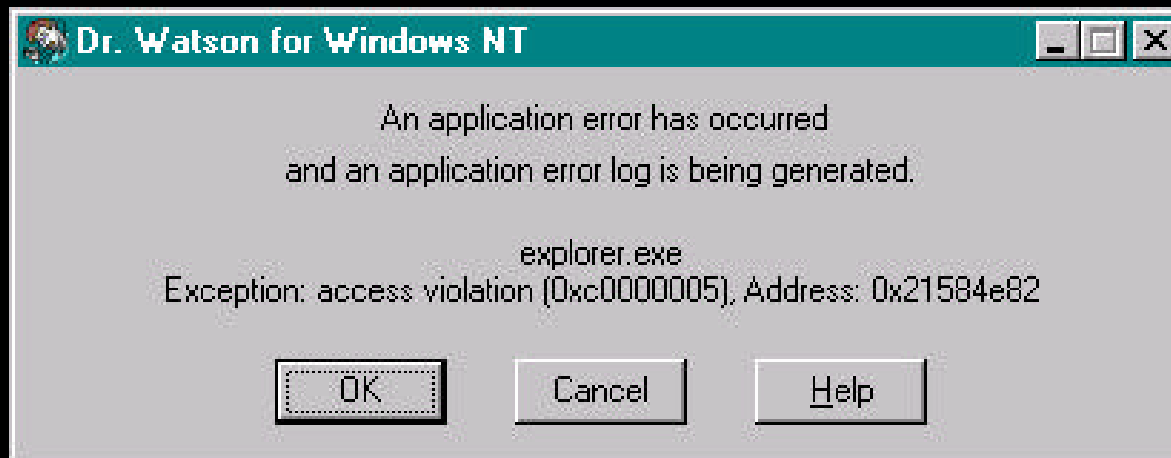
City     State

Next>

# Durable

## Is the product solid and reliable?

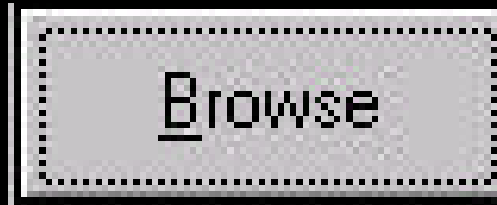
- ◆ How often does the software crash under normal use?
- ◆ How catastrophic are the crashes?
- ◆ How does the software hold up under non-ordinary use?



# Thoughtful

Does the product anticipate the users needs?

- ◆ Goes beyond testing if the program meets the specification
  - How useful and friendly is the program?
  - Is there anything missing that could improve the program?





# Testing With The Model

## ◆ Level 1: Awareness

- Testers who consider the model while testing will notice where these qualities lack
- Provide feedback to developers via defect reports

## ◆ Level 2: Reporting

- Provides feedback to managers
- Gives more attention to the subjective qualities
- Put feedback into numeric form that is easy for managers to digest

# Putting A Number To A Feeling

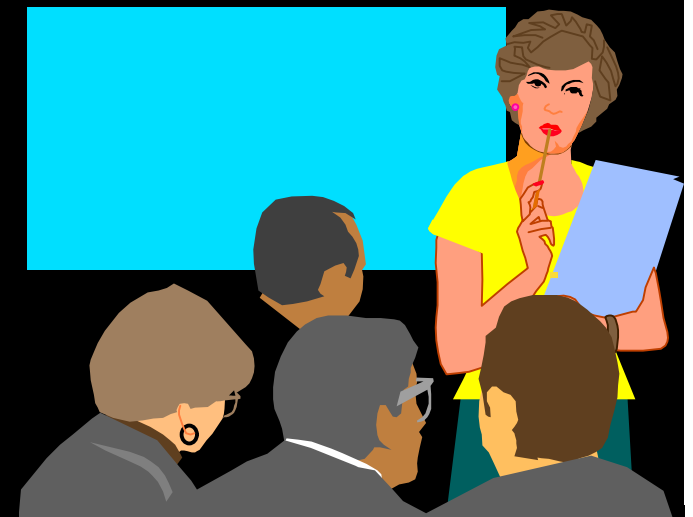
- ◆ Agree on a common scale
  - 1: The worst I've ever seen
  - 2: Worse than average
  - 3: About the same as other applications I've used
  - 4: Better than average
  - 5: The best I've ever seen
- ◆ Rate each category for each build tested

Build	Intuitive	Consistent	Efficient	Durable	Thoughtful
555	2	4	3	3	3
556	2	4	3	3	3
557	2	4	4	4	4
558	3	4	4	2	4
559	3	4	5	4	4

# Taking the ICED T Plunge

## ◆ Kick-off Meeting

- Involve all groups responsible for the product
- Make everyone aware of what the subjective qualities are
- Get buy-in from each group
- Agree on criteria for shipping



# Final Thoughts

- ◆ Customers' opinion is what matters most
  - This will be a subjective opinion
- ◆ Testers need to subjectively evaluate the software
  - Use the ICED T Model to go beyond the numbers and look at the total quality of the software

# Using The "ICED T" Model For Testing Subjective Software Qualities

Andy Roth  
Rational Software  
aroth@rational.com

---

## INTRODUCTION

Quality software. That is what we are seeking. While this is clearly a goal of any software tester or quality engineer, what exactly is the definition of quality software? Part of the answer is easy. There are many aspects of software that we can test and measure and to which we can assign a number. Some examples are how often the software crashes, how long it takes to complete a given task, or how much memory is being used. We can also look at how many of our tests pass and how many fail. While these quantifiable measures are important, they do not provide a complete picture of software quality. There are other more qualitative aspects of the software that also need to be considered.

By their very nature, these subjective aspects of software quality are harder to define and measure. Even so, as software testers we need to be aware of these aspects. We need to know how to evaluate them in a given application. Last, but certainly not least, we need to provide worthwhile feedback about them to management and development. The "ICED T" Model provides a structure for testing and reporting on subjective measures of software quality.

Before going into the specifics of the model, it is first necessary to look at the relationship between objective and subjective measures. By their definition, objective measures are those that are absolute and tend to be quantifiable (yes/no; on/off; 2,3,4...). Subjective measures, as the name implies, are subject to interpretation. These are the softer, more qualitative measures, which can vary from person to person, and even for the same person from one time to the next.

While these measures seem to be mutually exclusive, they are not entirely so. Because subjective measures are based on a person's experience, anything that affects that experience can affect their subjective evaluation. Take a glass of iced tea as an example. An objective evaluation of it includes such things as its temperature, the amount of sugar, the type of tea, and the number of ice cubes. These are all definite qualities of the beverage, and there is little room for argument about them. A subjective evaluation of the tea includes whether it is too hot or too cold, too sweet or not sweet enough, or most importantly, whether or not it is good. For the same glass of tea, there could be as many answers to these questions as there are people who drink it. Each person's subjective evaluation takes into account all the objective aspects of the tea, and looks at them through the "lens" of that person's own experience and preference.

Likewise when measuring the quality of software, the objective aspects of the software influence a person's subjective evaluation. For example, the objective measure of the time it takes for a given task to complete will influence the subjective measure of whether the application seems fast or slow. One person could subjectively consider the application to be fast and another person could subjectively consider the same application to be slow, even though the objective measure of time it takes to complete the task is the same for both persons. Software testers consider all the objective aspects of the application, and use these along with their own knowledge and experience to answer the key question of "How good is this software?".

Why is this important? When all is said and done, it is the customer's opinion of the software's quality that matters most. This opinion will be a subjective one. As such, we as testers need to also evaluate the software subjectively. Unfortunately, subjective measures do not lend themselves to being quantified or put into a graph or a report. This causes them to be left out of the

software quality reports which are given to those who make the product decisions. Focus tends to be on objective measures, like number of tests that pass versus the number that fail, or the number of known defects, while the more important question of "Is this quality software?" is overlooked. This emphasis on objectivity can lead people to believe that software that passes tests and has a low number of known defects is good software. As testers, we know this is not always the case. Customers using the software know this too.

---

## "ICED T" MODEL DEFINITION

The question of whether the software is good or not is too broad to be of much use. This is where the "ICED T" Model comes into play. It breaks down the overall subjective measure of the software's quality into five separate groups of related measures. These groups are **I**ntuitive, **C**onsistent, **E**fficient, **D**urable, and **T**houghtful.

<b>"ICED T" MODEL</b>	
<b>I</b>	NTUITIVE - Does the use of this product make sense?
<b>C</b>	ONSISTENT - Does the product operate in a uniform manner?
<b>E</b>	FFICIENT - Is the product quick and swift to use?
<b>D</b>	URABLE - Is the product solid and reliable?
<b>T</b>	HOUGHTFUL - Does the product anticipate the users needs?

Categorizing these measures has two main benefits. First, by calling out each group separately, it enables the tester to focus on all aspects of software quality, thus giving the tester a broader perspective. Second, it allows the tester to give more worthwhile feedback to developers and managers. Instead of simply saying "This software is no good", the tester can say *how* the software is not good. This allows the developers and managers to focus their attention more precisely on how to better the product. Let's examine each category more closely.

The first category looks at the aspects of software that determine whether or not it is **intuitive**. It answers the question "Does the use of this software make sense?" Things to look for when testing a software's intuitiveness include determining whether an action produces a logical result. This does not only mean that the application produces the result it is supposed to, but goes beyond that to look at whether that result really makes sense. Also, is that result what the user would expect? For example, a user pressing a button labeled "Back" expects the application to return to the previous screen. If some other screen appears, even if by design, this might not be intuitive.

Another aspect that affects an application's intuitiveness is whether or not the software's design implies its use. In other words, can users look at a screen and understand how they are supposed to use the interface and what actions should result? Such things as the layout of controls and the amount of on-screen information can affect intuitiveness.

The **consistency** of an application applies to whether it operates in a uniform manner. It can be looked at on three different levels:

- The first level is whether the application is consistent with itself across time. That means that if a user does an action today and does the same action tomorrow with all else being equal, they will get the same results.
- The second level is whether the application is consistent with itself throughout its user interface (UI). Are there standards for design and function that are followed throughout the product?

- The third level is whether the application is consistent with other applications. This can apply to other applications from the same company, as well as to general standards for the application's operating system. For example, if the application runs in Windows, and does not use "Ctrl-X" for Cut, then the application probably has a consistency problem.

The next category of subjective software qualities deals with the software's **efficiency**. Testing this category determines whether the application is quick and swift to use. Efficiency can be broken down into two areas: efficiency of the interface and efficiency of the code. The first area covers navigation. For instance, the number of keystrokes and mouse clicks it takes the user to accomplish a given task. Questions to ask when evaluating this are:

- Are shortcuts and hot-keys utilized effectively?
- Can any redundancies be eliminated?
- Is the application designed so that it requires a minimal amount of action and time to navigate?

Efficiency of the code deals with response time - in other words, how quickly the application responds to the user's input or how long it takes to complete a task. This should be evaluated in terms of quality and context. As such, it is very important to have a setup that is similar to that of the customer. For example, if you are testing a web application that most users will access via a dial-up modem connection, if you test over a T1 network connection, you may not get an accurate feel for how efficient the code is.

The subjective evaluation of efficiency is very much influenced by objective measures, perhaps even more so than any other category. It is very important when evaluating efficiency, to not only determine the time it takes to accomplish a goal using the software, but to also determine whether users will find this time to be acceptable in the context in which they use the software.

The **Durable** category covers software reliability. Such objective measures as crash rate and mean time to defect (MTTD) can affect this subjective evaluation, but it is the overall feeling of solidity that is of concern to this model. There are three aspects of durability to be evaluated for a given piece of software. First, how often does the software crash under normal use? If a certain function causes a crash one time, most users would probably consider it a fluke. If the function causes a crash regularly however, the application would be considered of poor quality. Second, how catastrophic are the crashes? Does the user lose data or was the program designed to auto-save data? Was the exception trapped? Can the application be restarted without restarting the OS or rebooting the machine?

The third aspect of durability deals with how well the software holds up under non-ordinary use. Hammers are not designed for putting in screws, but people will use them for that. Likewise, users will attempt things with the software that it was never designed to do. Testers need to imagine what these uses would be, try them out, and see what happens. Even if the application does not work that way, it should still not cause undue problems for the user who tries it.

The final category of subjective software qualities is perhaps the most subjective of all. It looks at how **thoughtful** the software is. This category also has the most overlap with the other categories. When evaluating an application's thoughtfulness, the tester determines if the product provides the users with everything that they need. Is anything missing from the program that would make it better?

This is an important thing to consider because it goes beyond just testing whether the program meets its specifications and examines whether the program is as useful and as friendly as it could be. A great example of a thoughtful addition to a program is the "Browse" button on a file access dialog. Without this button, users could still specify the file they wanted, and the program would work. By adding this button though, the program becomes much more thoughtful (not to mention efficient) by giving users an additional way to get to the file that may be easier for them.

A good way to evaluate an application's thoughtfulness is to adopt a "use case" based method of testing. Thus, instead of making sure that each feature does what it is designed to do, use a combination of features to accomplish something that a customer might actually do. With this mindset, you will soon think of better ways that the software could help users accomplish their goals.

---

## USING THE MODEL

Now that the "ICED T" Model has been explained, the important question becomes "How can we use this model to improve software quality?" For testers, there are two levels of implementation. The first is to be aware of what these subjective qualities are. Testers who consider this model while they test are more likely to notice when the software lacks any of these subjective qualities. Testers can then provide feedback to the developer so that the product can be improved.

One way to do this is to write a defect report. This defect report must explain why the specific quality is lacking. It is also very helpful to offer suggestions on how to improve it. For example, just saying a given dialog is not intuitive won't lead to a fix. Instead, a well written defect report explaining what is not intuitive about the dialog and how it could be improved will allow the developer to make meaningful improvements.

Depending on the people involved though, this may not be enough to really change how things are done. In such situations, the second level of implementation, providing feedback to management, can be used. This goes back to the idea that the decision makers on a given project will pay the most attention to reports and graphs when determining if the software is high enough quality and ready to ship. This is one reason why the "ICED T" Model is necessary. Thus in order to make the decision makers aware of quality aspects that don't lend themselves to being quantified, it becomes necessary to assign a numeric value to these aspects.

Of course, this task, by its very nature, is somewhat problematic. How exactly is one supposed to assign a number to a subjective feeling? This is where a tester's professional experience comes into play. An experienced tester should bring to the table the ability to look at a piece of software and make an informed opinion about the software's quality *as it is likely to be perceived by the software's end user*. This is one of the real values of having professional testers evaluate a piece of software instead of just using someone off the street. A software tester should be able to rate a piece of software on how intuitive, consistent, efficient, durable, and thoughtful it is, then assign a numeric value to these subjective ratings. For example, testers should have enough experience using various pieces of software to make an informed judgement about how intuitive the software they are testing is.

For these ratings to be meaningful across testers and across time, a scale should be set in place that gives guidance on what the numbers mean. If not, one tester might use "3" to mean one thing while another tester might use "3" to mean something quite different. The specifics of the scale are not important, just that everyone is using the same scale. For example, a possible scale might be:

- 1: The worst I've ever seen
- 2: Worse than average
- 3: About the same as other applications I've used
- 4: Better than average
- 5: The best I've ever seen

Once the scale is put into place, testers can rate each of the categories in the model for each build of the software that they test. An example report might look something like this:

Build	Intuitive	Consistent	Efficient	Durable	Thoughtful
-------	-----------	------------	-----------	---------	------------



555	2	4	3	3	3
556	2	4	3	3	3
557	2	4	4	4	4
558	3	4	4	2	4
559	3	4	5	4	4

The benefits of having testers report on these categories are three-fold. First, it encourages the tester to really think about the various subjective qualities. Second, it allows management to see the ratings of these qualities in a form that is easy to digest. Third, it allows the product to be compared over time - both through its development cycle and from one release to the next.

Use of the "ICED T" Model can also be beneficial to groups outside of test. When implementing the model, it is recommended to meet at the beginning of the project with all groups that are responsible for producing a quality product (which include, but are not necessarily limited to, development, test, and management). The first goal of this meeting is to make everyone aware of what the subjective software qualities are. Developers who consider the various subjective qualities when they are designing the software, are more likely to design quality software from the outset. Managers that are aware of the importance of these qualities are more likely to foster an environment in which the software will score highly in these measures. The second goal of the meeting should be to get buy-in from all those involved that these subjective qualities are important enough to warrant action when low scores arise. At the meeting, hammer out the definition of the scoring model, how the reporting will be done, and what criteria there are for shipping the final product.

---

#### FINAL THOUGHTS

When all is said and done, the customers' evaluation of an application's quality is what is most important. When they form this evaluation, they probably won't make a list of all the defects they find, or time how long it takes to accomplish a task. Instead, they will use the software to accomplish whatever it is they bought the software for in the first place, and through this use, they will form an opinion of how good the software is. This opinion will be subjective. As testers, an important part of our job is to do whatever we can to ensure that this opinion is a favorable one. That means that we need to go beyond our objective tests and measures, and consider the subjective quality of software.

The "ICED T" Model is a tool that will take these subjective measures and put them into a form that they can more easily be tested. Looking at an application in terms of its intuitiveness, consistency, efficiency, durability, and thoughtfulness will help the tester and product team consider the broader picture of the software's quality. This allows the tester to give important feedback to developers, managers, and others responsible for producing a quality product. By making these subjective qualities an integral part of testing, we can improve the software quality for whom it matters most: the customer. Quality software. That is what we are seeking.

## **ANDY ROTH**

---

Andy Roth is a quality engineer at Rational Software. He has a BS degree from Penn State University and an M.Ed. in Educational Computing Technologies from the University of North Carolina at Greensboro. Prior to coming to Rational in 1997, Andy worked in a variety of roles in the software industry including consulting, product marketing, system integration, functional verification, and international homologation testing.

Currently, Andy works with a team that tests Rational's PerformanceStudio product, a tool used for load and stress testing database and Web servers. Testing a product that is used by testers has given Andy an interesting perspective of software quality issues.