BOOK DRAFT—DEVELOPING SOFTWARE TEST CASES

Monday, September 23, 2003 tcase07equiv2.draft02.doc

Ross Collard

rcollard@attglobal.net

Chapter 7
More About Equivalence

Making Assumptions in Equivalence

If you comfortably understand the role of assumptions, how to make them, and how to recognize and validate them, then skip the first exercise. This exercise primarily re-visits concepts introduced in the last chapter. Assumptions are so important, however, and often so difficult to recognize and validate, that I have included this exercise as a deliberate reinforcement. It is subtler and harder than the exercises in the last chapter.

Exercise 7.1: Reviewing the Date Test Case Checklist

(Allow 45 to 60 minutes for this exercise.)

The chief financial officer (CFO) of your organization has hired an outside consultant to develop a checklist of date-related test cases. The CFO expects testers to use this checklist routinely in testing dates in the organization's finance and accounting systems which carry a high degree of risk and liability. Financial systems are usually highly date-sensitive, and the consequences of incorrect date processing can be very high, as we learned with the year 2000 scare. Dates which are low in risk are exempted from testing with this checklist.

The CFO has asked you to review the consultant's checklist, which I have attached below. The consultant must have used equivalence to narrow the set of conditions to test, as the number of possible test cases is huge. But the consultant did not document the assumptions underlying this checklist.

Questions to Address

Please answer these questions:

1. What assumptions has the consultant made about the equivalence of date fields, date-based computations and decision-making? Since no one has explicitly stated them, you will need to "reverse engineer" and extract the assumptions which he has buried in the checklist. List the assumptions of equivalence in each of these areas:

   A. System date (I define these terms below under the heading: "Date Terminology".)

   B. Today's date

   C. Other dates

   D. Intervals

E. Transaction dates vs. stored database dates

F. Method of date generation (automatically generated or manually entered)

G. Date field data values

H. Time zones and calendars

I. Date formats (e.g., 9/15/05 vs. 15 September, 2005)

J. Transaction amounts

2. Do you think each of these assumptions is reasonable?
3. How would you change these assumptions, if at all?
4. How do your changes in the assumptions translate into changes to the test case checklist?

Date Terminology

We'll use the following terms in this exercise:

- Today's date, versus system date and transaction date.
    - Today's date: this is the actual date today in the "real world", according to a fresh copy of the morning newspaper.
    - System date: the date to which the test environment is set, i.e., the date which the test environment and the system being tested "thinks" is today's date. I am assuming that the testers have the capability to set the date in the test environment to a day other than today's date, so that they can test what would have happened in the past or will happen in the future.
    - Transaction date: the date of a transaction, normally the date on which it was created or last modified. A transaction such as a hotel reservation can contain more than one date as internal data fields, which we refer to as the from-date and the to-date, as well as the date on which the transaction was created or modified.
- Future date versus past date.
    - Future date: any date which is later than the system date (not today's date).
    - Past date: any date which is earlier than the system date.
- Date field versus components.
    - A component of a date is the day, month or year.

    - The term "field" refers to the whole date, which is the combination of all three components: the month, day and year.

- Dates which denote the beginning and end of intervals.

    - An interval is the duration between two dates.

    - From-date: in a transaction with two dates, such as a reservation, this is the date of the beginning of the interval.

    - To-date: in a transaction with two dates, such as a reservation, this is the date of the end of the reservation interval.

The Date Test Case Checklist

The consultant has recommended the following set of test cases for each moderate-to-high risk date field in the input transactions.

1. Transactions with Today's, Future and Past Dates

    1.1. Test the transactions with the system date set to today's date.

        1.1.1. Test the transaction with the date of the transaction set to the system date. Example: Today's date:12/31/05 (December 31, 2005); System date: 12/31/05 (the test environment date); Transaction date: 12/31/05 (the date of an incoming test transaction).

        1.1.2. Test the transaction with a future date, i.e., set to a representative value more than the system date. Example: Today's date: 12/31/05; System date: 12/31/05; Transaction date: 01/15/06.

        1.1.3. Test the transaction with a past date, i.e., set to a representative value less than the system date. Example: Today's date: 12/31/05; System date: 12/31/05; Transaction date: 09/01/05.

        1.1.4. Test multi-date input transactions (e.g., transactions with both a from-date and a to-date, where the from-date should not be after the to-date.)

            1.1.4.1.From-date prior to the to-date. Example: Today's date: 12/31/05; System date: 12/31/05; Transaction from-date: 01/12/06; Transaction to-date: 01/18/06.

            1.1.4.2.From-date equal to the to-date. Example: Today's date: 12/31/05; System date: 12/31/05; Transaction from-date: 10/03/05; Transaction to-date: 10/03/05.

1.1.4.3.From-date after the to-date. Example: Today's date: 12/31/05; System date: 12/31/05; Transaction from-date: 10/22/05; Transaction to-date: 10/21/05.

1.1.4.4.From-date prior to the to-date, spread across the century boundary (i.e., before and after the year 2000).

1.2. Test the transactions with the system date set to a value other than today's date (either before or after).

1.2.1. Test the transaction with the date of the transaction set to the system date. Example: Today's date: 11/23/05 (the real date); System date: 12/31/05 (the test environment date); Transaction date: 12/31/05.

2. Date Errors

2.1. Test the transactions with invalid dates.

2.1.1. Day is not numeric.

2.1.2. Day is not in the range 1 - 31 (or 1 - 30 for June, September, etc.).

2.1.3. Month is not numeric.

2.1.4. Month is not in the range 1 - 12.

2.1.5. Year is not numeric.

2.1.6. The day and month components are individually valid but the combination is invalid, e.g., the testers should test one of the following conditions:

2.1.6.1.The day exceeds 31 in August.

2.1.6.2.The day exceeds 28 in February, in a non-leap year.

2.1.6.3.The day exceeds 29 in February, in a leap year.

2.2. Missing date in a field where the system requires the date; it is not optional.

2.2.1. Blank date. (The user enters blanks into the entire date field: month, day and year.)

2.2.2. Null date. (The user has entered nothing into the date field, but presses the Enter key or clicks the mouse to indicate that the date field has been entered.)

Answer to Exercise 7.1

I have listed the main assumptions below which the consultant has embedded in the date test case checklist. (This is not a complete list of assumptions, for brevity, but it contains the major ones)

Chapter 7
More About Equivalence - Answer to Exercise 7.1

A. System Date

All system dates are equivalent to each other, in the sense that any date which is set in the test environment is the same way as any other

- The day of the week is not relevant. The consultant has assumed that business week days (Monday through Friday), weekend days and holidays are equivalent to each other. In other words, we expect the systems to work the same way on Saturdays as they do on Tuesdays.

- Special or unusual dates, such as February 29, 2004 or December 31, 2003, are equivalent to "routine dates" and thus do not matter. The premise is that systems work the same way on these special days as on any other day.

- The origin of transactions in different time zones does not matter, although their times and dates may be converted to a common basis, i.e., to the same time zone. (The first few hours of Monday in Europe are still part of Sunday in the U.S.A., so that some European Monday transactions are considered Sunday transactions for U.S. accounting purposes.)

- The consultant has assumed that any daylight savings adjustments to transaction dates are not relevant.

- There is no such thing as an out-of-range date—no dates are too far into the past or the future to be valid.

- The system date may need to be set to a past date (e.g. yesterday's date) for testing. An observer could inquire: "I see the need to set the system date to today's date or a future date for testing purposes, but why would we want to set it to a past date? In some systems, there is a need to go back occasionally and re-process prior work loads on the original dates.

B. Today's Date

The one exception to the statement above that: "all system dates are considered equivalent to each other", is when the system date is set to today's date (i.e., the real date today).

All values of today's date (the actual date today) are considered equivalent to each other. In other words, it does not matter on which (real) day the test cases are executed. It could be back in the year 1800, if they had computers then, or the year 2100 (assuming they can remember what computers were).

- The consultant does not assume that the systems and transactions work the same way when the system date is set to today's date, as they work when the system date is set a value other than today's date.

- The consultant assumes that we do not need to test future and past transaction dates, as well as a transaction date equal to the system date, when the system date is not set to today's date. In other words, if we run the test cases in category 1. (1.1.1, 1.1.2, 1.1.3, 1.1.4, and 1.2.1) of the checklist, we do not need test cases 1.2.2, 1.2.3 and 1.2.4 below:

  1.2 Test the transaction with the system date set to a value other than today's date.

  1.2.1 Test the transaction with the date of the transaction set to the system date.

  1.2.2 Test the transaction with a future date, i.e., set to a typical representative value more than the system date.

  1.2.3 Test the transaction with a past date, i.e., set to a typical representative value less than the system date.

  1.2.4 Test a multi-date input transaction.

  (I omitted these three test cases, 1.2.2 through 1.2.4, from the prior list of date test cases.)

C. Other Dates

- All future dates are equivalent to each other, no matter how far they extend into the future. (I.e., tomorrow is equivalent to January 1st in the year 2 million AD.)
- All past dates are equivalent to each other, no matter how far they extend into the past.
- All from-dates are equivalent to each other.
- All to-dates are equivalent to each other. Note: an exception to this assumption and also the prior one about the from-dates is the turn-of-the-century processing. If these two assumptions about the equivalence of from-dates and to-dates are correct, then test case 1.1.4.4.4 (test a multi-date input transaction where the from-date is prior to the to-date, across the century change) would be the same as 1.1.4.1 (test a multi-date input transaction where the from-date is prior to the to-date, without a century change).

D. Intervals

All intervals are equivalent.

- We measure long intervals in years or decades, for example, in a pension system. For a person who started work in 1985 and is going to retire in 2022, an interval of 37 years is used to calculate his or her pension. We measure short intervals in minutes, seconds, or milliseconds, such as the time for an e-mail message to be transmitted from as server to a workstation. Short interval computations often have to be more precise than long ones. If we calculate someone's time on the job as 17.42 years versus 17.41 years, it will not affect her pension. If we measure the delay in a telecom network, by contrast, as 100

milliseconds instead of 200, it could make a big difference to equipment upgrade decisions. The premise here is that short intervals are equivalent to long intervals.

- All intervals of days between the from-date and the to-date are considered equivalent, provided that the to-date is after the from-date. The systems treat a large interval (e.g., years) in the same way as a small interval (e.g., a few days).

- All intervals of days between the from-date and the to-date are considered equivalent, provided that the to-date is before the from-date.

E. Transaction Dates vs. Stored Dates

A transaction date and a stored date with the same value are equivalent: if a test case works for a date stored in a database we assume it works for a date in an incoming transaction, and vice versa.

F. Method of Date Generation

It is irrelevant whether a system automatically generate dates or users manually enter them—these are equivalent. Based on this assumption, the date processing does not need to be tested for:

- Automatic date generation: where the system generates the date, to verify that it inserted the correct date into the date field.

- Manual override: if the system fills the date entry automatically, to verify the users can review and manually override it if necessary.

- Prohibition of manual override: to verify that unauthorized persons cannot override the pre-set date where the system prohibits this.

G. Date Field Data Values

- All non-numeric characters in date fields are equivalent. In other words, the consultant expects the system to react the same way if the data entered into the day component (or the month or year component) of the date are asterisks, dashes, slashes, question marks, or other non-numeric characters.

- Any numeric but out-of-range value entered into the day component is equivalent to any other one (e.g., a value not in the range of 1 to 31 for the day). In other words, if the system handles a value of zero in the day component correctly, and rejects it with the appropriate error message, it will behave the same way with any other out-of-range value, such as -1, +48,333 or -188.

- The same point also applies to the month component, for numeric values not the range of 1 to 12.

- Within an allowable range, any integer numeric value in a date component is equivalent to any non-integer numeric value. (E.g., the value of 3 for the day component is equivalent to the value of 22.487.)

- All invalid combinations of month, day and year are equivalent. In other words, if the system works a certain way with a date entered on a transaction of 6/31 (i.e., it rejects this input combination), it will work the same way with 2/29/2002 or 10/32/1966 or any other invalid combination.

- A date containing all blanks is equivalent to one containing blanks in only one component (e.g., the month component), or blanks in any combination of two components.

H. Time Zones and Calendars

- Time zones are equivalent. (For example, when it is 10.00 pm in Chicago on one day, it is 5.00 am in Paris on the next day, at the same instant in "real" time. The assumption is that this difference is time zones is irrelevant in testing.)

- The traditional Western calendar is being used, which contains both BC and AD years. It does not matter whether dates are BC (before Christ) or AD (anno domini)—they are equivalent. We covered this assumption already in earlier points, where I stated that all system dates are considered equivalent to each other, and all today's dates are considered equivalent to each other. I want to draw out this point about BC / AD equivalence so that it is apparent.

- If the system uses only the Western calendar, not inconsistent ones like the Muslim or the Mayan calendar (neither of which have 12 months), then the question of equivalence across calendars is irrelevant.

I. Date Formats

All date formats are equivalent.

- Alternatively, the consultant has assumed that all dates are in one consistent format only, e.g., U.S. dates in MMDDYY format. European dates (DDMMYY), and dates which not based on the Julian calendar (for example, some dates in Muslim countries are based on a lunar calendar) are assumed to be outside the scope of these testing efforts.

- Dates in MMDDYYY (with four digits for the year) are assumed to be equivalent to MMDDYY formats, in the sense that if a system works correctly when tested with dates

in either YY or YYYY format for the year component, we do not need to run another test suite with dates in the other YY or YYYY form.

- If there are multiple date formats used in a system, the statement that all date formats are equivalent assumes that the conversion processes back and forward among the formats are prefect (bug free).

J. Transaction Amounts

All transaction amounts are equivalent. In other words, in our test strategy we do not care if a transaction has a financial value of $250 million, $250 or 25 cents. In reality, our exposure from mis-handling the first transaction probably will be much higher than the 25 cents one. If we decide to subdivide transactions by amount, we may have to consider currency conversions. Having a different, high-risk test strategy for transactions with values greater than one million makes less sense for transactions denominated in Italian liras than in U.S. dollars.

What other assumptions can you identify, in addition to this list? It is scary when you realize how many unrecognized assumptions are buried in this simple date test case checklist.

Exercise 7.2: Checking the Reasonableness of the Assumptions

(Allow 30 to 45 minutes for this exercise.)

Equivalence testing is only as good as the assumptions we make. We need assumptions when we do not have all the facts, which is most of the time. I am not claiming that the above list of approximately 20 assumptions is correct; but that the consultant has buried these assumptions in the date test case checklist. Are these assumptions about dates reasonable? I will leave this for you to decide. Think about your own personal experience with date-related financial systems, such as the ones which manage your bank accounts and credit cards. Ask yourself:

- First, was the reverse engineering done correctly? Have we adequately identified all the significant assumptions which are buried in the date test case checklist?
- Based on my experience, do I feel comfortable with each assumption listed above?
- Are there any assumptions where I really have no experience and thus no opinion except blind superstition?
    - Are all these assumptions significant? (If an assumption has low consequences, it is not worth the effort to check it out.)
    - For the ones which are significant, how practically can I check them out? Are there any sources of information which I can access?

- This issue in addressed in the section of this book entitled: "Recognizing and Checking Assumptions", in Chapter tbd.

Lessons Learned from this Exercise

- Without knowing the hidden assumptions of equivalence behind the date test checklist, the consultant's first draft checklist probably seems reasonably acceptable at first glance, especially to non-testers or testers who do not think critically.
- It is difficult and perhaps impossible to critique the checklist and find its flaws (either in terms of unnecessary duplication or risky gaps in the test coverage), without knowing the premises behind it.
- The author of the checklist was sloppy in not providing his or assumptions, forcing us to re-engineer from the checklist to derive the assumptions. We should routinely attach assumptions to suites of test cases.
- If the author did not list his assumptions, the chances are that he does not have a list. In other words, he or she probably has not recognized and carefully thought his own assumptions through, and used them to self-critique his work before giving the test case checklist to others.
- If the assumptions are unstated, we can go back to the author asap and ask him to provide them.
- If the author cannot provide the assumptions, we need to derive them from the date test checklist, assuming that we care enough about the quality of the checklist to justify the effort.
- Reverse-engineering and deriving assumptions is often aggravating. It requires careful thinking and can be time consuming.
- But the rewards, in terms of the improvement to the set of test cases, can be very satisfying.

Exercise 7.3: Calculating the Test Coverage

(Allow 20 to 30 minutes for this exercise.)

Based on the assumptions of equivalence in the last exercise, calculate the coverage ratio of the date test checklist. This is the ratio of the number of equivalence test cases to the number required for an exhaustive test of every way in which dates could be used. As an interim step to finding this ratio, first calculate approximately how many equivalence test cases we need. For this exercise, ignore stored dates and consider only the dates in incoming transactions.

Chapter 7
More About Equivalence - Answer to Exercise 7.3

We have some background data which will help. A recent, statistically valid sampling of representative incoming transactions found the following demographic mix and transaction volumes:

1. 77% of all transactions do not have any internal data fields which contain dates, though each of these transactions does have a transaction date (the date of its creation).
2. 23% of all transactions have two internal data fields which contain dates, the from-date and the to-date.

   2.1. Of these transactions, 19.5% contain a from-date which is less than the to-date.

   2.2. 3.25% contain a from-date which is equal to the to-date.

   2.3. 0.25% contain a from-date which is greater than the to-date (usually in error).
3. 95% of transactions are processed on the same day they are created, i.e., the transaction date is equal to the system date of the environment.

   3.1. 4% of transactions are processed after the day on which they are created.

   3.2. Transactions which are modified after they are created are 1% of all transactions. Of these, all are modified one or more days after they were created. 95% are processed on the same day they are modified.

   3.3. No transactions can be processed before they are created.
4. 1% of all incoming transactions have an error in one or more date fields (e.g., month greater than 12).
5. The organization processes an average of 10 million transactions per day.
6. The number of different ways in which transactions could be processed is indefinitely large, including all possible transactions occurring on all possible days during the active lives of the processing systems. I therefore assume that an exhaustive test is a volume test containing one day's worth of transactions (10 million).

Answer to Exercise 7.3

Equivalence Class	Number of Test Cases Needed	
	Equivalence	Full Test

Equivalence Class	Number of Test Cases Needed	
	Equivalence	Full Test
1. Transactions with Current, Future and Past Dates		
1.1 Test transactions with the system date set to today's date.		
Test transactions with the transaction date set to the system date.	1	7,365,000[1]
Test the transaction with future dates	1	77,000
Test transactions with past dates	1	308,000
Test multi-date input transactions		
From-date prior to the to-date	1	1,950,000
From-date equal to the to-date	1	325,000
From-date after the to-date.	1	25,000
From-date prior to the to-date, spread across the century boundary.	1	?[2]
1.2 Test transactions with the system date set to a value other than today's date.	1	500,000
2. Date Data Entry Errors		
Test transactions with invalid dates.		100,000
Day is not numeric.	1	
Day is not in the range 1 - 31(or 1 - 30 for June, September, etc.)	1	
Month is not numeric	1	

[1] I have calculated this as (95% x 77%) of 10 million.

[2] The information is not available to compute this number. We will assume it is very small.

Chapter 7
More About Equivalence - Answer to Exercise 7.3

Equivalence Class	Number of Test Cases Needed	
	Equivalence	Full Test
Month is not in the range 1 - 12	1	
Year is not numeric.	1	
The day and month combination is invalid.	1	
The day exceeds 31 in August	-	
The day exceeds 28 in February in a non-leap year.	-	
The day exceeds 29 in February, in a leap year.	-	
Totals	14	10,000,000

The test coverage ratio here is 14 test cases to a full day's volume of 10 million. Small ratios like this are what testers like to see. Will these 14 test cases catch all the bugs that 10 million test cases would? Probably not. There are many arcane combinations of dates in the live data which the set of 14 test cases does not include. Most transactions contain other data fields in addition to dates. For example, order transactions contain order quantities and customer names and addresses, while bank deposits contain customer account numbers and deposit amounts. These situations are not included in the 14 date-related test cases, but we will cover them in the volume test.

Nevertheless, this exercise illustrates the power of equivalence to dramatically shrink the number of test cases needed. We generally catch the majority of bugs with the equivalence test cases, if our assumptions and analysis are sound. We can expect to find perhaps 95% of the bugs which a full volume test would uncover. (Though this figure of 95% is just a guestimate unless we are willing to run volume tests to see.) Not a bad yield for only 14 test cases.

In addition, bugs might be found with the 14 test cases which we cannot find with a volume test, since the volume test only processes transactions from one day. The set of 14 equivalence test cases, by contrast, includes test cases where the system date in the test environment is set to different dates, both today's date and other than today's date.

If the risk justifies it, we also can perform a more comprehensive and conservative analysis of the combinations of conditions, and develop a list of maybe 100 equivalence test cases. This can push the percentage of bugs found much higher than the 95% figure mentioned above.

Equivalence Problems and How to Resolve Them

In this section we will address a few complications (which, when we face them, aren't really so bad):

- How to handle uncertainties about the equivalence classes: how to define them and decide which items belong in each one.
- How the complexity of system features and capabilities influences equivalence.
- How far to subdivide equivalence classes into more granular equivalence classes, and when to stop.
- How to handle the multiple dimensions of equivalence: items which are equivalent from one perspective may not be from another.
- How to test combinations of equivalence classes.

Uncertainties about the Equivalence Classes

Equivalence is nice in theory, but we often experience a problem in its practical application. Sometimes the equivalence classes are quite clear and their boundaries are distinct, and then there is no problem. Other times, we struggle to define the equivalence classes. Two different people on the same day, or the same person on two different days, develop different classes.

The first question is what attributes or characteristics we want to apply equivalence to. For example, let's say we want to test a system which handles the sales of drinking glasses—everything from their catalog presentation, pricing, selection and ordering, inventory management, packing and shipping, to billing. Our firm stocks and sells thousands of glasses, and we don't want to have to test every type of sale for every type of glass.

To develop equivalence classes, do we categorize the glasses by the type of liquid they are intended to hold (wine vs. water, etc.), the type of material used to manufacture them (lead crystal, etc.), whether they are hand-blown or machine-made, the glass size, weight or volume, the glass color, the list price, the wholesale price, the quantities in which we sell the glasses (singly, packaged by the dozen, or in bulk to restaurants), whether the glasses are imported or domestic, or the vendors who supply us with the glasses? Do we categorize the geographical area of the customer's shipping location, the taxes due on glass sales, or the customer's method of payment?

There are other attributes of a glass which we don't care about, such as the degree to which a particular glass deviates from perfect circularity, and its opacity. However, the manufacturer's testing department will be very interested in these attributes. And if we sell glasses, we probably also sell related accessories like bottled water, corkscrews and ice trays. How do we fit those into our suite of test cases?

Chapter 7
More About Equivalence - Equivalence Problems and How to Resolve Them

What if we are not confident that we have identified the right characteristics and equivalence classes? In the case of the drinking glasses, assistance hopefully is available. The marketing people may already have thought through how to coherently organize the merchandise for presentation on a Web site and in a catalog. The database administrator already has grouped items together and has a sense of which information about glasses is pertinent to the system we are testing.

An accompanying source of uncertainty is the decision rules which determine whether an item is a member of a particular equivalence class. The rules are not always black and white. When is a wine glass not a wine glass but a water glass or a parfait glass? (Never heard of a parfait glass for serving desserts? Testers don't get out much, do they?)

If there is significant uncertainty about equivalence, and if the risk justifies the effort, we test more than one sample test case from an equivalence class. The higher the uncertainty and risk, the more test cases are logically justified. We can sample randomly across the class, or instead choose test cases according to a particular selection technique like boundary value (BV). The BV approach to test case design is described later in this book, in Chapter tbd: pay particular attention to the section on fuzzy boundaries.

Equivalence and Complexity

The relative richness and complexity of a system's capabilities influence their equivalence: the more complex a feature is, the more the number of equivalence classes or subdivisions needed for that capability. Let's say that we want to test a hand-held computing device. Hundreds of our co-workers will be using this device outdoors in all sorts of working conditions, ranging from very hot and dry desert areas to cold, moist and rainy places. The device will replace an existing one with which it is backwards-compatible, i.e., the new device should be able to do everything that the existing one does. The only differences are that the new device offers some new features and enhancements, and the new device works faster and is lighter.

We will use equivalence in testing the backwards compatibility of the new device. One capability of the device is the "on" button. This button has a simple function; it turns the device on, and the button either works or it doesn't. Our assessment of the equivalence of the "on" buttons on the new and existing devices comes down to these questions: (1) Are the buttons expected to work the same way? (2) Has the same electro-mechanical switching component been used for the new and existing buttons? (3) Are the environmental conditions equivalent in which the devices are used? For example, if the existing device is used in a hot and dry desert can we assume the new one is used in the same environment? (4) Are the environmental conditions equivalent, in the sense that if the 'on" button works in the desert it also works in a cold and moist place, or vice versa?

The first question in turn leads to two more questions: (5) Do we take the same action (or equivalent actions) to turn on both the existing and new devices? (6) Do we expect the same outcome to result from our action, i.e., the devise is on and ready to use? The answer to both

these questions is yes, so the expected behavior of the "on" button should be equivalent. There may be differences in the behavior, for example, the new device may power up much faster, but we'd deem those differences immaterial.

We also know that the same electro-mechanical switching component has been used for the new and existing devices. Users have switched on hundreds of thousands of copies of the existing device, tens of millions of times. They have reported no problems with switching on the existing devices in hot and dry areas, but have experienced problems in cold, moist places. In other words, we can't assume equivalent behavior across different physical environments—if the new device's button works OK in the desert, then we can't assume it will also work OK in the rain.

So we have two equivalence classes and thus two test cases for the "on" button: try it in a hot, dry climate, and in a wet, moist one. We'd actually take a small sample of the new devices and try switching them on a few times under hot and dry conditions. (In theory we'd need to test only one—equivalence states that if one "on" button works they all will work), but out of prudence we'll try a few.) We'd try another sample under cold and moist conditions. Because the risk of failure is higher, we'd use a larger sample, and try turning on each one more times, under the latter conditions.

Another capability built into the device is a Web site search via wireless and the Internet. The search capability is much more rich and complex than the "on" button. It uses proprietary software, and the diversity of searches which the device can perform in theory is indefinitely large. (Practically, our willingness to be patient for long searches places an upper limit n the number of possible searches.) Let's say that the search software has been extensively upgraded for the new device, and we do not know anything about its internal workings. We'd have to use equivalence unless we wanted to test every conceivable search. The question is, how can we apply it? Can we assume equivalence across all searches, in other words, if the device works correctly for any one search then it will work for any search? We'd be unwilling to make this assumption, as there are simply too many variations among Web searches.

So we'd have to group the Web searches together (i.e., form equivalence classes), based on their similarities. We might decide to cluster searches together based on these factors: (a) whether we have an exact URL (universal resource locator) or IP (Internet protocol) address, a specific, narrow topic, or a general topic, (b) whether we expect most of the matching sites to be listed in the directories of the major search engines or to have private IP addresses, and so on. Two equivalence classes sufficed for the "on" button, but regardless of how we decide to group the searches, we'd probably end up with considerably more than two equivalence classes for the search capability.

"Infinite" Subdivision

With the term "infinite subdivision", I am not talking about an ambitious housing developer who is trying to squeeze more and more condominiums onto a small piece of land. Instead, I want to

address how to know when to stop subdividing each equivalence class into subsets of smaller equivalence classes.

Consider, for example, determining how to test the editing of an input character string which can have a variable length. As a test strategy, we decide to subdivide the universe of all possible strings into short strings with 5 characters or less, or long ones. Then we decide to subdivide the short strings further, into ones with lengths of 0 characters, 1 character, 2, 3 and so on. We subdivide the set of all two-character strings into the alpha-only ones, ones containing non-alpha only, and ones which mix alphabetic with non-alpha characters. Next we subdivide the alpha-only strings into upper-case only (e.g., "AA"), lower case only ("bb") or mixed ("xQ"). We separate the mixed ones into upper-case-first versus upper-case last ("Aa" versus "aA"), and then separate the upper-case-first ones into same-letter versus different letter ("Aa" versus "Aq".). I could go on... From one equivalence class (character strings with 5 character strings or less), we could subdivide into thousands of distinct equivalence classes if we wanted to.

How much subdivision of the equivalence classes is enough—what is the stopping rule? The answer to this question is experiential—we need to feel for the bottom as we go. As we subdivide we need to consider whether the extra work entailed by the new level of subdivision (by having several test cases instead of one), is justified. If, for example, we are confident that the value of the extra information produced by multiple test cases is minimal, and the cost of developing and running these test cases is not trivial, then we have subdivided too far. If the system or feature is already available to test, we can check this by running one or two of the more detailed test cases, selected from the subdivided set, in experimental mode.

When we review a test strategy, a useful way to check whether the equivalence classes are adequately broken down is to take a few of them, fragment them further as a trial and decide whether the new test cases are worthwhile. By trying to subdivide to a level which is one too deep, we can discover when we have hit the bottom.

The Multiple Dimensions of Equivalence

The next line sounds like a Zen riddle: When are equivalent items not equivalent? Two (or more) items which are equivalent from one angle may not be from another. An item is anything we are interested in examining, such as a feature, Web page, e-mail message, test case or personal computer.

As an example, compare these two items which are both books. Whether they are equivalent or not depends on which characteristic we consider:

Characteristic of the Item	Item A	Item B	Equivalent?	Relevant to us?[3]
1. Type of item	Book	Book	Yes	Yes
2. Topic	Science fiction	Science fiction	Yes	Yes
3. Price	$19.95	$19.55	Yes[4]	Yes
4. Color of cover	Red	Blue	No	No
5. Weight	2.5 pounds	1.8 pounds	Yes	No[5]
6. Date published	March 2001	May 2003	No	No
7. Title	"Achieving Zero Bugs"	"How I Flew to the Moon"	Yes[6]	Yes
8. Availability	In stock	Out of stock	No	Yes
9. Review	"Way cool"	"Groovy, man"	Yes	Yes
10. Font	Times Roman 12 point	Arial 10 point	Yes	No[7]

Using the Information

Can we profitably use the information in this table which compares the two books? If so, how? Let's say we work in a book shop and have a customer who wants to purchase a book. The customer is interested in this title: "How I Flew to the Moon", but currently it is out of stock. Will the customer accept a substitute book which is in stock? This way, everyone will be happy – the customer has a book to read, and we have made a sale.

---

[3]This column notes whether we care if items A and B are equivalent.

[4] These prices are close enough ($19.95 versus $19.55), to be considered equivalent. The same is true of the weights (1.8 versus 2.5 pounds).

[5] The potential readers are relatively insensitive to the book's weight. If it is a good read, they do not care too much about the weight of a book.

[6]These book titles are considered equivalent, in the sense that a typical science fiction fan has an equal interest in reading either one.

[7] As long as the font is readable and not distracting, the readers do not care what the specific font is. So the Times Roman and Ariel fonts are equivalent—and thus irrelevant. Would we still think that the font is irrelevant if the choice was between the Times Roman 18 point and the Ariel 6 point fonts?

According to the data in the above table, the customer does not care about the book color, the date published or the weight, so we can ignore these characteristics. Of course, if the customer has to carry the book on a long journey and the weight of a substitute book is 50 pounds, not 1.8 pounds, the customer will certainly care. A weight of 2.5 pounds is equivalent to 1.8 pounds for the customer's purpose, but not 50 pounds. The equivalence of the book weight depends on how the book will be used. A reader who carries the book in a backpack is probably more sensitive to weight than someone who parks the book on a bedside table.

According to the table comparing items (i.e., books) A and B, the customer cares about these characteristics: (a) the type of item (the customer will not accept a tennis racquet or a music CD instead of a book), (b) the topic (he only reads science fiction), (c) possibly the price (he has a budget which he can't change), but he is insensitive to a 50 cents difference (d) the title (he does not want to buy a book he has already read), (e) the enthusiasm of the review, and (f) the availability (it is no good if we substitute one out-of-stock book with another which we can't supply either).

Equivalence is not always a simple black-and-white, yes-or-no decision as I've indicated in the table. The customer might be highly sensitive to some factors, such as the enthusiasm of the book review, and buying another copy of a book he already has. By contrast, the book weight and price are minor issues within certain ranges of tolerance, and major issues outside those ranges. In this case, we show the customer another equivalent science fiction book, entitled "Achieving Zero Bugs", tell him that it is an equally fanciful, heroic and enjoyable tale but with a sad ending. (The evil aliens who rule the world blame the testers for the bugs which got away. Thank goodness this tale is only fiction!) We make the sale, as the books are equivalent for our purposes.

Can you identify other situations where these two books are not equivalent? What about the book replenishment or re-ordering process? If we are out of stock for "How I Flew to the Moon" and overstocked with copies of "Achieving Zero Bugs", it makes no sense to order more copies of the second one.

A Software Test Case Example

In testing a Web site, we are interested in the compatibility of the user interface, the browser and operating system (OS) used by a visitor to the site. We develop two test cases, one of which initiates a query to the Web server through the Internet, from a Linux machine which uses Internet Explorer and a T1 line. The other test case initiates a similar query from a Windows machine with the Navigator browser and a dial-up modem. I'll call this part of the test cases, from the user to the database, the front end.

These two test cases both access the same record (which is stored in the same Oracle database, and this database reside on the same database server). Both test cases perform the same computation and are expected to deliver the same outcome. I'll call this part of the test cases, the

data retrieval and computation, the back end. We can show this information in a table of equivalencies:

Characteristic of the Query	Query A	Query B	Equivalent?	Relevant?
Version of test case	Linux/Oracle test case	Windows/Oracle test case	Maybe[8]	Yes
User interface	Linux/Oracle user interface	Windows/Oracle user interface	Maybe	Yes
Database availability	Linux/Oracle database	Windows/Oracle database	Yes[9]	Yes
Value of stored data	Linux/Oracle stored data	Windows/Oracle stored data	Yes	Yes
Query outcome	Linux/Oracle result	Windows/Oracle result	Yes	Yes

Let's now change the scenario. In a new situation, we are testing a software package which can only run on a Macintosh computer (so there is only one front end), but can access both DB2 and Oracle remote databases through the Internet (there's two different back ends). These two databases are supposed to contain comparable data, so that the same query should yield the same answer from both databases.

We develop two new test cases for this situation. On the front end, both these test cases initiate queries through the Internet from one common Macintosh machine. On the back end, these two test cases access the two different databases. How does our test strategy change? Well, let's review the new table of equivalencies:

[8] We would like these factors to be equivalent, but do not know if they are before we run the test cases. We also need to define what equivalence means in these circumstances—for example, does it mean that we can expect an exact pixel-by-pixel match of the screen images, or only that the look-and-feel of the two browser displays are similar?

[9] We are confident that the database availability, stored data values and query results are equivalent, because in each of these circumstances items A and B are one and the same thing. We will still run the test cases to make sure.

Characteristic of the Query	Query C	Query D	Equivalent?	Relevant?
Version of test case	Mac/DB2 test case	Mac/Oracle test case	Maybe	Yes
User interface	Mac/DB2 user interface	Mac/Oracle user interface	Yes	Yes
Database availability	Mac/DB2 database	Mac/Oracle database	Maybe	Yes
Value of stored data	Mac/DB2 stored data	Mac/Oracle stored data	Maybe	Yes
Query answer	Mac/DB2 result	Mac/Oracle result	Maybe	Yes

The question was how our test strategy changes when the scenario changes. Based on the new table of equivalencies for queries C and D, we'd spend considerably less time testing the compatibility of the user interface, though we would still want to check it, and much more time testing the back-end database compatibility.

These conclusions about the equivalence of the test cases may seem intuitively obvious, and we may reach the conclusions in a flash without bothering to analyze the situation. In other words, isn't this laborious comparison of the factors overkill? Yes—and no. We do not want to get bogged down in trivia—we can't afford to waste our time. On the other hand, without analysis we miss subtle relationships which change our opinion about the equivalence of test cases.

The guideline I use is this. If the factors which have a significant influence on equivalence number more than half a dozen, if the interrelationships among these factors are unknown, if we have little prior experience in the domain, and the risk associated with the area is not minor, then I check my first intuitive sense with as more thorough, pen-and-paper analysis.

Exercise 7.4: The Dimensions of Equivalence

(Allow 20 to 30 minutes for this exercise.)

Background Description

A team of automobile insurance claims analysts carry hand-held computers to accident sites and local garages, where they use these computers to assess the costs of damages. The computers contain custom-built application software. The users (i.e., the claims analysts) love the rich

functionality and convenience of the software, but hate its slow performance. The team's manager has decided to increase the team's productivity and morale by upgrading to new hand-held computers.

We have been asked to test the new devices before they go into use. As part of our test planning, we need to analyze the equivalence of the old and new devices, based on this information:

- The same manufacturer that supplies the new computers as the existing ones.
- The keyboard and touch-pad display panel of the new model are unchanged, except that the display contains a new icon to warn if the battery charge is low.
- Each computer has a plug-in portable printer which can be used to print contracts and releases on the spot. The same family of printers and the same print drivers (software components which handle printer communications), can be used on the new computer.
- The new computers will run the same operating system as before, but the operating system will be upgraded to a newer version which is considered to be faster and cleaner (less bugs).
- Each new computer contains a processor chip which is twice the speed of the chip in the existing computers.
- The old and new processor chips are part of the same chip family, which means they have similar architectures and run the same firmware (basic operating instructions).
- Each new computer also contains twice the amount of memory as the older model.
- Functionally, we expect the new computer to be able to run application software the same way as the old one, though there could be minor exceptions. The same claims processing application software will run on the new computers as the old.
- The batteries are physically plug-compatible, in other words, we expect the same battery to work with either an old or new computer. At the same time that the manufacturer releases the new computer, it is making a new battery available which allows the computer to run for twice as long as the current batteries. While this new battery is plug-compatible with the old computer, the manufacturer recommends that we do not use the new battery in the old computer because of the high amount of heat dissipated.

Chapter 7
More About Equivalence - Exercise 7.4: The Dimensions of Equivalence

Questions

1. In which of these areas is it safe to assume equivalence, so the areas do not need intense testing on the new hand-held computer? In which areas can we not assume equivalence?

Area of System	Old-and-New Equivalence? (Yes / no / unsure)	Relevant to Us? (Critical / important/minor / irrelevant)
a. Password control		
b. Look-and-feel of user interface		
c. Data entry through the touchpad		
d. Data entry through the keyboard		
e. Display of the date and time		
f. Database search of automobile repair costs		
g. Retrieval of customer record and policy data		
h. Computation of accident costs		
i. Application of local taxes		
j. Timing, out-of-synch or race conditions[10]		
k. Response time		
l. Battery life		
m. Print capability		
n. Internet access		
o. Other (describe)		

[10] Race conditions can happen when the same software is ported to a faster or slower hardware platform. See the discussion of feature interactions in Chapter tbd.

2. Are there any other important factors which have been omitted from the list? If so, what are they? Add these factors as extra rows at the bottom of the table.

3. Based on the information you have, which areas of difference justify more test cases, or fewer?

4. What other information, if any, would you like to have before proceeding?

5. Based on your answers to the above, what is your recommended test strategy for the new computer?

Combinations of Equivalence Classes

In many situations, there are a multitude of equivalence classes which can interact and influence the testing. Imagine, in the earlier example where reminder notices are generated for overdue accounts, that we have three different types of customers: Finnish, Estonian and Latvian. Because of different regulations in these three countries, the processes used to generate reminder notices are distinct for each country. In other words, we can't assume that Finnish, Estonian and Latvian customers are equivalent.

Lets also assume that our firm applies interest charges on overdue balances, on three levels based on the customer's business volume: a customer who buys over $1,000 of merchandise per year is charged no interest, one who buys between $500 and 41,000 is charged 5%, and under $500 the rate is 10%. Suddenly we have nine combinations to test.

If we throw in a few more variables, the number of combinations of equivalence classes quickly becomes overwhelming. We'll address the topic of testing combinations of items in a later section of this book, entitled: (tbd).