1. Essential Testing Methods

Box approach: White and black boxes

White box testing means that the tester knows the code (internal data structures plus algorithms) corresponding to the test cases .

Examples: API testing, code coverage, conditions testing.

Black box testing (Functional testing)

It treats the software as a black box and tests the functionality at high level as specified in the requirement without a priori or posterior knowledge of the code. Only the input-outputs and specification are known.

Examples: User interface, data entry to and retrieval from, and data manipulation of a database.

2. Examples of test case construction for unit testing and functional testing

source code: currencya.php

```
<?php
//define variables
$num1 = $_GET["TXT1"];
$code1 = $_GET["SELECT1"];
$code2 = $_GET["SELECT2"];

$carray = array(
'EUR'=>'Euro',
'HKD'=>'Hong Kong dollar',
'GBP'=>'Pound sterling',
'CNY'=>'Renminbi',
'USD'=>'United States dollar'
);
$crate = array(
'USD/EUR'=>'0.7139',
'USD/GBP'=>'0.6196',
'USD/CNY'=>'6.5051',
'USD/HKD'=>'7.7755',
'USD/USD'=>'1.0000'
);

//build input amount textbox
function txtbox($num){
$txt = "<input id='txt".$num."' name='TXT".$num."' type='text' size='10' value="</>"."\n";
return $txt;
}

//build option tags in select. make use of array
function selectbox($array){
$txt = '';
foreach($array as $key=>$value){
$value = htmlentities($value);
$txt .= "<option value='".$key."'>".
$value."</option>"."\n";
};
return $txt;
}

function result($rate, $array){
global $num1, $code1,$code2;

//check errors or range
if(is_null($num1)) {return 'Please enter an amount in the input box.';}
if(!is_numeric($num1)) {return '"'.$num1.'" in input box is not a number. Please enter a number.';}
if($num1<0) {return '"'.$num1.'" in input box is negative. Please enter a positive number.';}

if($code1==null) {return 'Please choose a country below the From-box.';}
if($code2==null) {return 'Please choose a country below To.';}

if($code1==$code2) {
$tmp = $num1.' ('.$array[$code1].')='.$num1.' ('.$array[$code2].')';
return $tmp;
}

//find the conversion rates
$USD_c1 = 'USD/'.$code1;
$USD_c2 = 'USD/'.$code2;

$cnt = 0; $rate1=1; $rate2=1;

$rate1=1.0/$rate[$USD_c1];
$rate2=$rate[$USD_c2];

//do the conversion
$amount = $rate1*$rate2*$num1;
$amount = number_format($amount, 4, '.', '');

//report back as a string
$tmp = $num1.' ('.$array[$code1].')='.$amount.' ('.$array[$code2].')';
return $tmp;
}
?>

<!-- use the action of form to call back Ex2a.php -->
<form name="form2" METHOD="GET" ACTION="EX2a.php">
<table cellspacing="1" cellpadding="1">
<tr>
<td>
<h3>Currency Converter:</h3>
<br> &nbsp;&nbsp;
<br> &nbsp;&nbsp;
<br> &nbsp;&nbsp;
</td>

<td>

<table>
<tr>
<td>
<table>
<tr><td>
&nbsp;&nbsp;From:<?php echo txtbox('1'); ?>
</td></tr>
<tr><td>
<select name='SELECT1' size='4' >
<?php echo selectbox($carray); ?>
</select>
</td></tr>
</table>
</td>

<td>
<table>
<tr><td>
&nbsp;&nbsp;To:&nbsp;&nbsp;
</td></tr>
<tr><td>
<select name='SELECT2' size='4' >
```

```
108. <?php echo selectbox($carray); ?>
109. </select>
110. </td></tr>
111. </table>
112. </td>
113. </tr>
114. </table>
115. </td>
116.
117. <td>
118. <table>
119. <tr>
120. <td>
121. <input type="submit" value="Convert" /><br />
122. </td>
123. </tr>
124. </table>
125. </td>
126. </tr>
127. </table>
128. Answer: <?php echo result($crate,$carray); ?>
129. </form>
130.
131. --------------------------------------------------
132.
133. Source code: Ex2b.html
134. <html>
135. <head></head>
136. <body>
137. <H1 style="text-align:left;">Easy123 Travel Ltd.</H1>
138. <?php
139. include('currency2b.php');
140. ?>
141. </body>
142. </html>
```

Combinatorial analysis of the test cases in the currency converter example

1. Elementary cases: amount/to-currency/from-currency: Number of test cases > 2*5*5=50
2. The combined dimensions of the input-output space is huge.

Practical strategy:

- Binary tree analysis
- Pick the normal data range
- Pick the out-liners
- Pick the exceptional cases
- Include any critical test cases as supplied by the customers or obtained in the fields.

Final catch: Be realistic. Results of the combinatorial analysis indicate that if the complexity of a software is high, we cannot guarantee that a piece of the software be always correct. We cannot even guarantee that the specifications are correct because there exists no verification system that can verify every piece of a specification and its code in a program. Above all, we cannot even guarantee that a verification system is itself correct because it is a vicious circle on its own. However, as long as the software is running within the normal range and the software is accumulatively improved and fully tested, then the reliability of a software can reach an acceptable level without incurring unrecoverable economic losses or severe losses of lives for everyday deployment.

Examples

a. Unit testing:

Test cases for the logic in line numbers (white box) 42, 43, 44, 46, 47 and 49

Line	Input data	Expected value = =actual value? PASS: FAIL
42	empty/y/z	Please enter an amount.
43	abc/y/z	Amount is not a number.
44	-3/y/z	Please enter a positive number

46	any/no selection/any	Choose a country below the input box
47	any/any/no selection	Choose a country below To
49	numeric/x/x	Should give answer 1 to 1.

The collection of test cases in their respective categories are called test suites.

b. Functional testing (black box):

Example test cases

Precondition	Execution steps	Expected results==actual results? PASS:FAIL
1 /HKD/USD	Enter 1 in amount box	
	Click HKD in selection 1	
	Click USD in selection 2	
	Click convert	Front page reappears and 1HKD=0.1286USD
10/HKD/USD	Enter 10, ditto	Front page reappears and 1HKD=1.2861USD
0.1/HKD/USD	Enter 0.1, ditto	Front page reappears and 1HKD=0.0129USD