The Grey-Box Software Testing Method *The Functionality of Software Quality* Appendix

Jerry Huth, Solid Step Software, Inc

I. Case Study: Using Solid Step Test with C++/.NET

This appendix accompanies the paper titled *The Grey-Box Software Testing Method: The Functionality of Software Quality*. It includes code fragments to illustrate how Solid Step Test (Sst) was used to cover the interfaces of har*GIS Field Information Systems' "Truckmap" software, a large GUI application that uses the C++/.NET platform.

Each subsection of this appendix corresponds to a subsection of section V of the main paper. For instance, subsection C of this appendix corresponds to section V, subsection C of the main paper.

A. Object Registration

During testcase playback, messages are sent to the GUI objects to replay the individual GUI events (or actions) that are saved in the serial test log. Examples of objects receiving these messages include classes derived from CFrameWnd, CWinApp, CView, etc. When these objects are created, they are registered with the SstPlayer (the capture/playback engine) to give each object a unique identifier. For instance, Truckmap's CView-derived class is CTruckmapView, and the following Line of Code (LOC) was inserted in its constructor (ctor) to register it with the Sst libraries:

```
CTruckmapView::CTruckmapView()
{
  ...
  SstPlayer.RegisterType( this, "TruckmapView") ;
```

And the following LOC was added in the object's destructor (dtor) to unregister it:

```
CTruckmapView::~CTruckmapView()
{
  ...
  SstPlayer.Unregister( this) ;
```

All GUI objects that receive notifications from Sst are registered in a similar manner.

B. Menu Items

Menu items are typically covered by adding 1 LOC in the callback function for the menu item. For instance, in Truckmap, the "Pan Left" menu item is covered like this:

```
void CTruckmapView::OnPanleft()
{
  SstCmdTarget( this, ID_PANLEFT) ;
```

In this implementation, SstCmdTarget is a class, not just a function. This LOC instantiates an object of that class, whose ctor stores the data in the serial test log. During testcase playback, the OnPanLeft() function is triggered when the SstCmdTarget class' OnPlayback() function sends the ID_PANLEFT message to the CTruckmapView object.

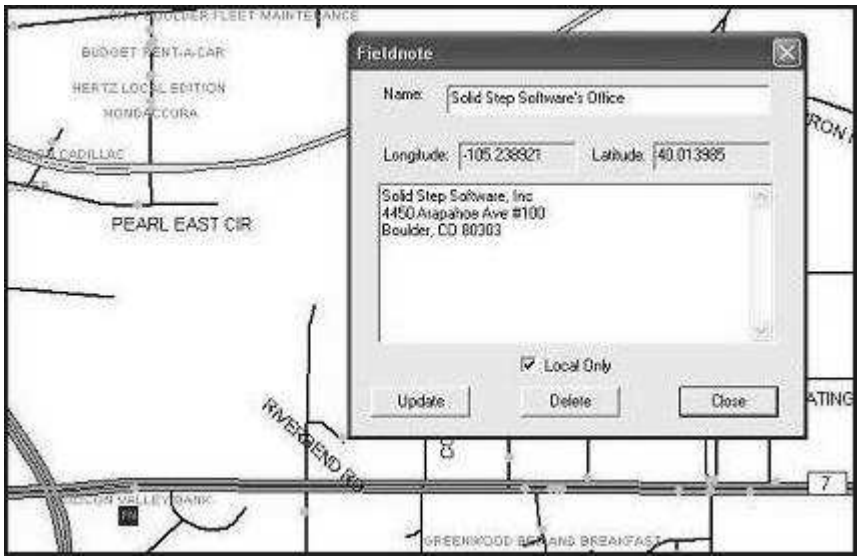

Fig. 1. Truckmap's Fieldnote Dialog

C. Keystrokes & Button Clicks

Keystrokes and button clicks are handled in a similar manner to menu items, i.e. by adding 1 LOC in the callback function:

```
void CTruckmapView::OnLButtonDown(UINT nFlags, CPoint point)
{
  SstLBD( this, nFlags, point) ;
```

As in the previous example, SstLBD is a class whose ctor saves the data in the serial test log.

D. Window Position and Size

To handle changes to a window's position or size, Solid Step Test provides the SstMoveWindow class, which saves a window's position and size in the serial test log. However, since the .NET framework can handle changes to a window's position and size automatically, these events were not represented explicitly in Truckmap's code.

In order to cover these events, Truckmap's main window's OnMove() and OnSize() virtual functions were overridden and LOC's added to use the SstMoveWindow class:

```
void CMainFrame::OnMove(int x, int y)
{
    CFrameWnd::OnMove(x, y);
    SstMoveWindow( this) ;
}

void CMainFrame::OnSize(UINT nType, int cx, int cy)
{
    CFrameWnd::OnSize(nType, cx, cy);
    SstMoveWindow( this) ;
}
```

E. Simple Dialog Boxes

Simple dialog boxes, which do little or no processing until the user closes it, are typically handled by adding 1 LOC in the ctor and dtor, and 1 or a couple LOC's in each callback function that can close the

dialog. The following example shows how the Fieldnote dialog is covered. This dialog, shown in Fig. 1, has three buttons that can close it: an “Update” button, a “Delete” button and a “Close” button.

```
ctor:
  CFieldnoteDlg::CFieldnoteDlg(CWnd* pParent /*=NULL*/)
          : CDialog(CFieldnoteDlg::IDD, pParent)
  {
    SstPlayer.RegisterType( this, "FieldnoteDlg") ;

dtor:
  CFieldnoteDlg::~CFieldnoteDlg()
  {
    SstPlayer.Unregister( this) ;

Update callback:
  void CFieldnoteDlg::OnUpdate()
  {
    SstSetDlgChecks( this,
                     IDC_CHECK1,
                     NULL) ;
    SstDlgCmd( this, IDC_UPDATE,
               IDC_EDIT,
               IDC_EDITLAT,
               IDC_EDITLON,
               IDC_EDIT1,
               NULL) ;

Delete callback:
  void CFieldnoteDlg::OnDelete()
  {
    SstDlgCmd( this, IDC_DELETE) ;

Close callback:
  void CFieldnoteDlg::OnClose()
  {
    SstDlgCmd( this, IDCANCEL) ;
```

Of particular interest is the “Update” callback function, which not only saves the event in the serial test log, but also saves the current values of the text boxes and checkboxes of the dialog in the test log. This is required because the “Update” callback function uses the values of those GUI elements to modify the internal state of the Fieldnote object, while all the other callback functions for this dialog simply react to the event regardless of the values of the individual GUI elements of the dialog.

F. Complex Dialog Boxes

More complex dialog boxes, which do a significant amount of processing as the user interacts with the individual GUI elements of the dialog, may need those GUI elements to be covered individually. For instance, Truckmap has a dialog named “CGNMgrDlg” (short for Geo Note Manager Dialog), which lists all the various kinds of geonotes the user has created. If the user clicks on a geonote in the list, the callback function for the list element fills in the other fields of the dialog with the specific values of that geonote:

```
  void CGNMgrDlg :: OnSelchangeMpsktlistCtr(NMHDR* pNMHDR,
                                            LRESULT* pResult)
```

To make this dialog reproducible, it was necessary to be able to record and play back the list selection actions. This was done by adding a new action class specific to this dialog, which when played back caused that callback function to be called:

```
class CMpSktMgrSelect : public SstListCtrlSelect
{
  ...
  virtual void OnPlayback() {
    CMPSKTMGRSELECT_BASE_CLASS::OnPlayback() ;
    CGNMgrDlg *dlg = (CGNMgrDlg *)
                        Mgr()->Get( m_MpSktName) ;
    if( dlg) {
      LRESULT result ;
      dlg->CallOnSelchangedFunction( NULL, &result) ;
    }
  }
```

The public function CallOnSelchangedFunction() was added to the CGNMgrDlg class since the OnSelchangeMpsktlistCtr() function was not public:

```
void CGNMgrDlg::CallOnSelchangedFunction(
                    NMHDR *pNMHDR,
                    LRESULT *pResult)
{
  OnSelchangeMpsktlistCtr( pNMHDR, pResult) ;
}
```

G. Toolbars

Toolbar move events present a similar problem as the window move/resize events discussed above, namely that toolbar move events were not explicitly represented in Truckmap's code because the .NET framework handled those events automatically. But unlike the case of window move/resize, the class that receives the events that need to be saved in the testcase also was not directly represented in Truckmap's code. So to handle toolbar move events, a subclass of the .NET toolbar was added to Truckmap, and the OnWindowPosChanged() virtual function overridden to handle the move events:

```
/*********************************************************************

        NAME: CTmToolBar

        AUTHOR: Jerry Huth

        ABSTRACT: This is a subclass of CToolBar that knows how to
        save move/resize events in the test log.

*********************************************************************/

class CTmToolBar : public CToolBar
{
public:
  DECLARE_DYNAMIC( CTmToolBar)
  CTmToolBar() ;
  ~CTmToolBar() ;
```

```
        //{{AFX_MSG(CTmToolBar)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()

  virtual void SaveSstDockState() ;
  afx_msg void OnWindowPosChanged(WINDOWPOS* lpwndpos);
};

void CTmToolBar::OnWindowPosChanged(WINDOWPOS* lpwndpos)
{
        CToolBar::OnWindowPosChanged(lpwndpos);
        SaveSstDockState() ;
}

void
CTmToolBar::SaveSstDockState()
{
  ((CMainFrame *) m_pDockSite)->SaveSstDockState() ;
}
```

The main window's toolbars were changed to use the new class:

```
class CMainFrame : public CFrameWnd
{
  ...
        CTmToolBar    m_wndToolBar ;
        CTmToolBar    m_wndRouteBar ;
        CTmToolBar    m_wndViewBar ;
        CTmToolBar    m_wndNavBar ;
        CTmToolBar    m_wndMiscBar ;
        CTmToolBar    m_wndGCBar ;
        CTmToolBar    m_wndSketchBar ;
  ...
```

And the SstDockState class was used to save the position of all the toolbars:

```
void CMainFrame::SaveSstDockState()
{
  SstDockState( this, &m_wndToolBar, &m_wndRouteBar, &m_wndViewBar,
                &m_wndNavBar, &m_wndMiscBar, &m_wndGCBar,
                &m_wndSketchBar, NULL) ;
}
```

H. AfxMessageBox

The AfxMessageBox facility provided by .NET, which gives the application programmer an easy way to interrupt the application and give the end user a simple message, or ask a simple question, could have been covered in a couple different ways. The development team considered implementing their own message box functionality, which would have allowed them to make the message boxes completely reproducible, i.e. to make the message boxes appear during testcase playback.

But another much simpler way to cover this message box functionality was tried, and it ultimately worked so well that there seemed to be no need to replace .NET's message box functionality. This approach involved using .NET's message box virtual function feature to avoid displaying the message boxes during testcase playback and instead simply return whatever value was returned during the original

run of the application. More specifically, the CWinApp DoMessageBox() function was overridden by this function:

```
/*********************************************************************

        NAME: CTRUCKMAPApp::DoMessageBox

        AUTHOR: Jerry Huth

        ABSTRACT: Override this function so that we can skip the msg
        boxes during testcase playback and instead just return what
        was returned during the original run of the application.

*********************************************************************/

int CTRUCKMAPApp::DoMessageBox(LPCTSTR lpszPrompt, UINT nType,
                               UINT nIDPrompt)
{
  bool can_show_msg_box = true ;
  bool should_save_msg_box_ret = true ;
  int retval = IDOK ;
  SstDoMessageBox *msgbox_action ;

  // Depending on what's desired, choose the default ret val. //
  if( nType & IDNO) {
    retval = IDNO ;
  }
  if( nType & IDCANCEL) {
    retval = IDCANCEL ;
  }

  // If we're playing back. //
  if( SstPlayer.PlaybackMode()) {

    // Don't show the msg box. //
    can_show_msg_box = false ;

    // Unless we're paused. //
    if( SstPlayer.IsPlaybackPaused()) {
      can_show_msg_box = true ;

      // In which case we shouldn't save the return value. //
      should_save_msg_box_ret = false ;
    }
  }

  // If not playing back, see if this is a msg that shouldn't have its //
  // return saved (for instance because the action that caused the msg //
  // box to be displayed isn't being saved).                           //
  else if( SstPlayer.DontSaveMsgBoxRetVal()) {
    should_save_msg_box_ret = false ;
  }

  // Show the msg box if desired. //
  if( can_show_msg_box) {
    retval = CWinApp::DoMessageBox(lpszPrompt, nType, nIDPrompt);
```

```
    // Save the return value if needed. //
    if( should_save_msg_box_ret) {
      SstDoMessageBox( retval, SstMfcGetRealStr( lpszPrompt)) ;
    }
  }

  // Else get the value to be returned. //
  else {
    msgbox_action = (SstDoMessageBox *)
                      SstPlayer.Peek( "SstDoMessageBox") ;
    if( msgbox_action) {
      retval = msgbox_action->GetRetVal() ;
      delete msgbox_action ;
    }
  }

  return( retval) ;
}
```

I. Simple Data Files

"Simple" data files, such as picture files that are used by a drawing program, are files that are only read and written at specific times, i.e. when the user directs the program to read or write the files. These files are covered by inserting a call to Solid Step's reproducibility libraries at the point where the file is read or written.

Although in general only input files need to be covered to enable reproducibility, in practice output files are often covered in the same manner to facilitate easy data verification. In both cases, during the original run of the program, Solid Step Test copies the file into the testcase's file bundle. Then during testcase playback, input files are read from the file bundle so the correct input file data is used by the application. Also during testcase playback, Solid Step Test redirects the output files to a special output file area and the output data is automatically verified against the output files stored in the testcase file bundle.

Although during testcase playback the application "thinks" it is reading input file data and writing output file data to and from the same file locations it was using during the original run of the program, Solid Step Test actually redirects the locations to files in the file bundle or in the special output file area. In this way, Solid Step Test lets the program run during testcase playback in what is essentially a "sandbox", so that it runs as it originally ran but without requiring the input files to be in their original locations and without causing the original output files to be overwritten.

Although Truckmap does not use simple input data files (it uses database files for all its data - see the "Database Files" section below), it nevertheless dumps the database fields to simple output files to make use of Solid Step's easy data verification functionality (see the "Data Verification" section below).

J. Database Files

In contrast to simple data files, database files are continually read and written as the application runs. Covering database files is done in essentially the same manner as simple data files, i.e. by adding a call to Solid Step's libraries at the place where the file is used in the application:

```
int GEONTDB::SetDatabase(CString TName)
{
        if(VCCEADO==NULL){
                VCCEADO = new cADOCE();}
        if( VCCEADO_DATABASE != TName) {
```

```
            VCCEADO_DATABASE = TName ;
            VCCEADO->DATABASE=SstReadDBFile( SstMfcGetRealStr( TName)) ;
          }
          return 1;
    }
```

Like simple data files, database files are copied into the file bundle. During testcase playback, however, another copy of the file is made - a “sandbox” version - which the program reads and writes as the program runs. This lets the program run during testcase playback in the same way that it ran during the original run of the application, but without reading or modifying the files from their original locations, and without modifying the files in the testcase file bundle - i.e. as noted above, the program runs in a sandbox provided by Solid Step’s reproducibility engine.

K. The Windows Registry

For applications that use the .NET platform, another very important interface that must be covered to enable reproducibility is the Windows registry. As with other interfaces, it is covered by adding calls into Solid Step’s libraries at the points where the Windows registry is accessed. This example shows how the SstReg class, which provides functions with profiles similar to the registry functions originally used in Truckmap, is used in place of those functions:

```
//open the specified key
if(SstReg.OpenKeyEx(INITIAL_KEY, m_KeyPath, 0, KEY_READ,
                    &phkResult)!=ERROR_SUCCESS)
{
        //if key can’t be created then set the error message
        m_RegistryLastErrorMessage.Format(_T(
            "Can not open %s key in the Registry"),m_KeyPath);
}
//if the key is opened then do following
else
{
        //local variable of size int used the read the
        //value from the registry
        unsigned long size_of_int = sizeof(int);

        //query int value from the specified registry path,
        //the value is read in the variable m_ObtainValue
        //if unable to read then set the error message
        if(SstReg.QueryValue(phkResult, m_Key, m_ObtainValue)
                          !=ERROR_SUCCESS)
                m_RegistryLastErrorMessage.Format(_T(
        "Can not query %s key from the registry"),m_Key);

        //close the key, and if can’t close it then
        //set the error message
        if(SstReg.CloseKey(phkResult)!=ERROR_SUCCESS)
                m_RegistryLastErrorMessage.Format(_T(
        "Can not close %s key in the registry"),m_Key);
}
```

In this way the data returned from the Windows registry is saved with the testcase and during testcase playback that saved data is returned to the application so that it runs in the same way that it ran during the original run of the application.

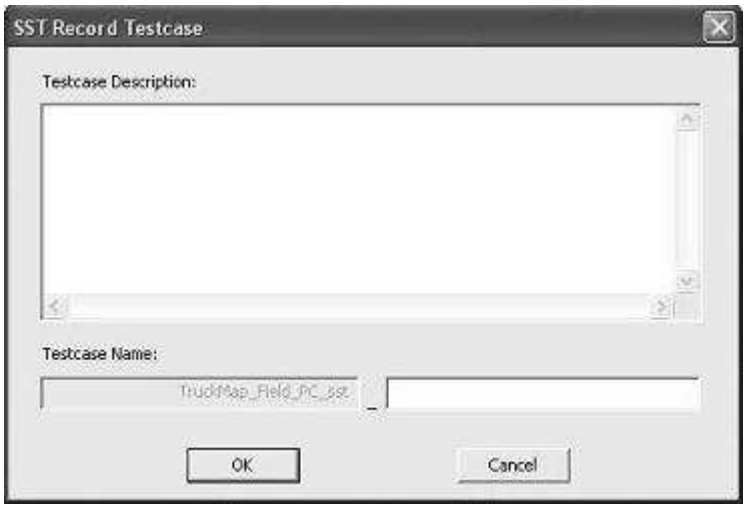

Fig. 2. Recording a Testcase

L. Testcase Creation

When the interfaces of the application are sufficiently covered by calls into Solid Step's libraries, then Solid Step Test saves all the data in a testcase as the application runs, and at any later time Solid Step's reproducibility engine can coordinate all the data to replay that exact run of the application. In this way, Solid Step Test provides an "Always On" functionality that works behind the scenes to create testcases whenever the application is run.

There are a number of ways that Solid Step Test can be used in an application. In the most likely scenario, Solid Step Test can be configured to overwrite the previous testcase if the application had exited normally with no apparent errors. If the application exited abnormally, however, then Solid Step Test could automatically save the testcase for later evaluation by the application programmers. In another scenario which may be useful in certain situations, for instance during trial testing periods, Solid Step Test could be configured to save all the testcases, or to save a random sampling of testcases.

Another very common scenario when testcases will need to be saved is when an application programmer/tester is creating a functionality or regression test suite. In Truckmap a special dialog was added for this purpose, as shown in Fig. 2, and it lets the user specify the testcase name and allows them to include a description of the functionality that the testcase is meant to verify.

M. Data Verification

Although Truckmap's output data is stored in a database file, since Solid Step Test includes an automatic data verification feature for regular (non-database) output files, code was added to dump Truckmap's data to regular files. Then when testcase playback is finished, Solid Step Test automatically verifies the data against the data stored with the testcase:

```
/*********************************************************************

        NAME: CSstPlayerDlg::OkCallback

        AUTHOR: Jerry Huth

        ABSTRACT: Do whatever the user wants to do, like save
        data to files.

*********************************************************************/
```

```
void
CSstPlayerDlg::OkCallback()
{
  SstStr dir = "test_data" ;
  SstStr file ;

  dir = CreateTestDataDir() ;

  if( SstIsChecked( IDC_SAVE_BOOKMARKS)) {

    // Dump the data. //
    SstDumpFileData( dir, "bookmarks.txt", file,
                     TmView().geoMapper.DumpBookMarkData,
                     SstIsChecked( IDC_INCLUDE_BOOKMARK_FLOATS)) ;

    // If we didn't dump all the data, do it again but don't //
    // tell SST about it.                                    //
    if( !SstIsChecked( IDC_INCLUDE_BOOKMARK_FLOATS)) {
      FILE *f = fopen( SstFile::CreatePath( dir, "bookmarks.all.txt"),
                                            "w") ;
      TmView().geoMapper.DumpBookMarkData( f, true) ;
    }
  }

  if( SstIsChecked( IDC_SAVE_FIELDNOTES)) {

    // Dump the data. //
    SstDumpFileData( dir, "fieldnotes.txt", file,
                     TmView().geoMapper.DumpFieldNoteData,
                     SstIsChecked( IDC_INCLUDE_FIELDNOTES_FLOATS)) ;

    // If we didn't dump all the data, do it again but don't //
    // tell SST about it.                                    //
    if( !SstIsChecked( IDC_INCLUDE_FIELDNOTES_FLOATS)) {
      FILE *f = fopen( SstFile::CreatePath( dir, "fieldnotes.all.txt"),
                                            "w") ;
      TmView().geoMapper.DumpFieldNoteData( f, true) ;
    }
  }
}
```

The dialog in Fig. 3 lets the user choose which data to dump out and also lets the user insert a "Pause" into the testcase event log.

This example also illustrates how some kinds of data, such as the floating point numbers that represent latitude/longitude values, may not always need to be saved with the testcase. This is useful since testcases whose purpose is to verify functionality unrelated to the exact floating point values might be unnecessarily brittle if the floating point numbers were always saved with the testcases.

N. Case Study Discussion

The focus of this project was to see how well Solid Step Software's new SQA approach would work in a large commercial application with an embedded GUI. So even though some parts of the Truckmap application were beyond the scope of this project, such as the GPS features, its very large GUI provided an excellent opportunity to explore how this approach would work in a large software system that uses a leading development platform such as .NET. As this experience has shown, Solid Step Test works very well in practice with a full-featured platform like this since by its nature the platform provides all

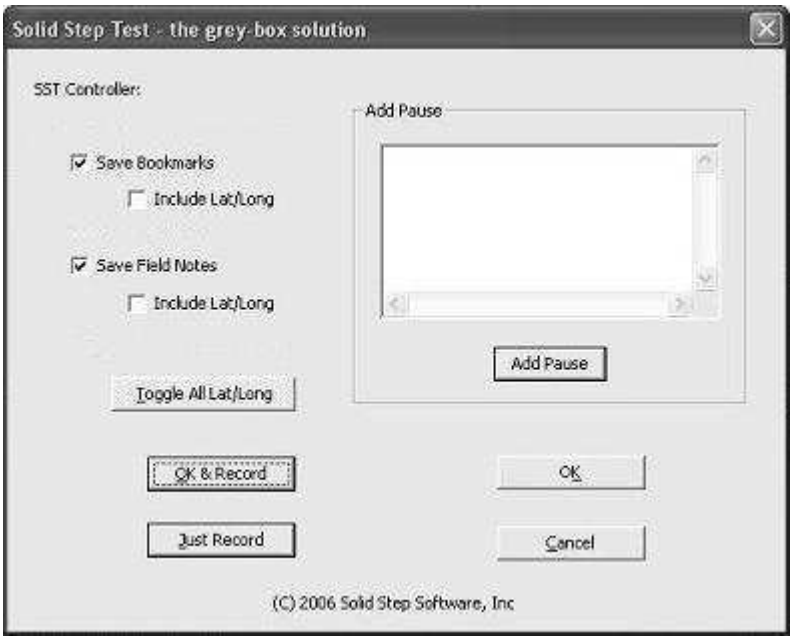

Fig. 3. Dumping Data to Output Files

the hooks, callbacks, etc, needed to allow the application programmer to insert the calls to Solid Step's libraries required to make the application reproducible.

For instance, although the interfaces that need to be covered by calls to the Solid Step libraries were usually readily available in Truckmap's code, in some cases they were not represented in Truckmap's existing code. But even in those cases, the interfaces were easily exposed, either by overriding virtual functions in existing subclasses, or by creating subclasses and then overriding the virtual functions, as was done to cover the toolbar events.

It is expected that other full-featured platforms, such as Sun's Java platform, or the X Windows platform, etc, would also work well with Solid Step Test since by their nature as full-featured platforms they would include all the hooks, callbacks, etc, required to properly expose any interfaces that need to be covered by calls to Solid Step's libraries.

Visit www.solidstep.com to download a demo of a Solid Step Test enabled application, learn more about Solid Step Software's new SQA solution, or to contact the author.