vApus v2

Dieter Vandroemme

March 12, 2012

Contents

1 Why? 2

2 Over Viewing the Fundaments (D) 2

2.1 Static GUI . . . 3
2.2 Managing Modules The Actual Framework . . . 4
2.3 Generated GUI . . . 5
2.3.1 Stresstesting Solution Explorer . . . 5
2.3.2 Tree Nodes . . . 5
2.3.3 Context Menu's . . . 6
2.4 Resource Management (Images) . . . 8
2.5 File Management (Automated Saving, Loading and Cloning . . . 9
2.6 Important Other Stuff . . . 9
2.6.1 Comparers and Extensions . . . 9
2.6.2 SpecialFolder . . . 10
2.6.3 ClipboardWrapper . . . 10
2.6.4 SocketWrapper . . . 10
2.6.5 SynchronizedContextWrapper . . . 11
2.6.6 Win32WindowMessageHandler . . . 11

3 Stresstesting with vApus v2 (DR) 11

3.1 What is Stresstesting? . . . 11
3.2 A New Approach in vApus v2: Standardized Stresstesting Model . . . 11
3.2.1 Lexing and Parsing: Rule Sets . . . 11
3.3 Lexing and Parsing in vApus v2 (D) . . . 11
3.3.1 Lexing . . . 11
3.3.2 Lexical Result . . . 11
3.3.3 Parsing . . . 11
3.4 The GUI for All This (and a Quick Note for Researchers) . . . 12
3.4.1 Parameters . . . 12
3.4.2 Connections . . . 12
3.4.3 Logs . . . 12
3.5 Stresstesting . . . 12
3.5.1 Configuring a Stresstest (and Some Best Practices) . . . 12
3.5.2 Starting a Stresstest and Handling Errors and Other Problems . . . 12
3.5.3 Saving Results . . . 12

4 Stresstesting with vApus v2: Explaining the Code (D) 12

4.1 Rule Sets . . . 12
4.1.1 Lexing . . . 12
4.1.2 Parsing . . . 12
4.2 Parameters . . . 12
4.3 Logs . . . 12
4.4 Connections . . . 12
4.4.1 IConnectionProxy and Programming Your Own Connection Proxy Class . . . 12
4.4.2 ConnectionProxyPool . . . 12
4.4.3 Free Coding . . . 12
4.4.4 Unit Testing . . . 12
4.5 Threading . . . 13
4.5.1 StresstestThreadPool . . . 13
4.6 StresstestResults . . . 13
4.7 Stresstesting . . . 13
4.7.1 Starting a Test . . . 13

4.7.2 Executing the Work 13
4.7.3 Displaying Results 13
4.7.4 Handling Errors and Other Problems 13
4.7.5 Stopping a Test 13

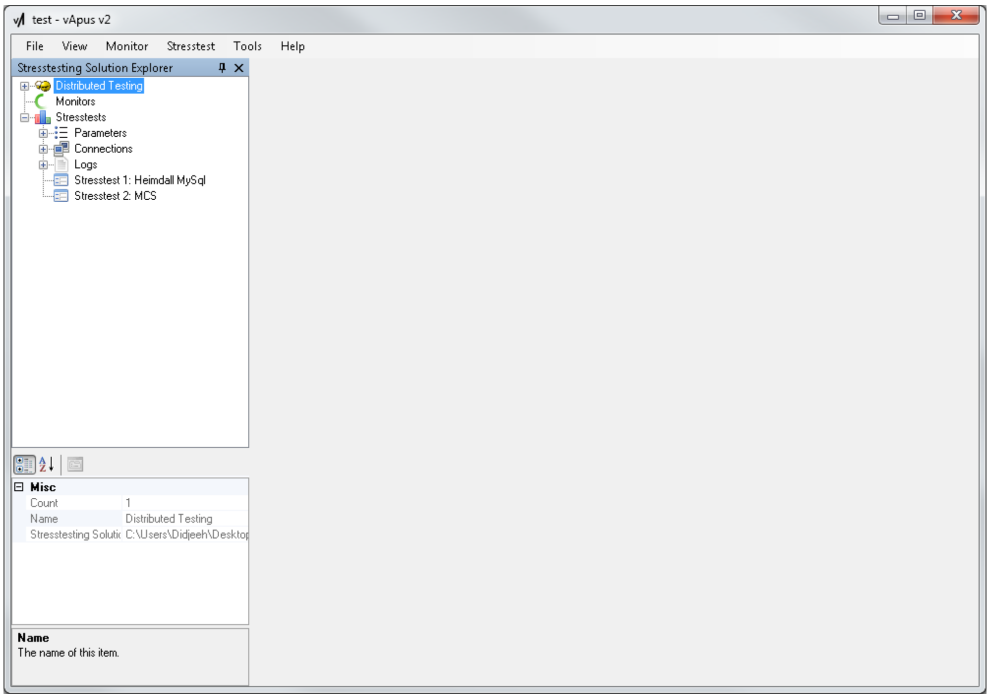
5 Distributed Testing (DR) 13
5.1 Setting Up a Test (and Some Best Practices) 13
5.1.1 Resource Pool 13
5.1.2 Tiles 13
5.1.3 vApus Jump Start (Experimental) 13
5.2 Starting a Test, Handling Errors and Handling Other Problems 13

6 Distributed Testing: Explaining the Code (D) 13

1 Why?

Why is a fairly good question. Why a second version? Well, it was time for a fresh start, erasing all made mistakes in the past by me and my predecessors. vApus v1 has its issues, it is bloated, not dynamic (adding new stresstests or new anything for that matter is a pain) and usability is lacking for the user and the developer. All this will be explained in the following sections, these will describe how to use vApus v2 and how it was built. That way it is a guide for both future developers and researchers. Sections particularly interesting for developers are indicated with a **(D)** in the title, for developers and researchers both with a **(DR)**.

2 Over Viewing the Fundaments (D)



Misc

Count	1
Name	Distributed Testing
Stresstesting Solutic	C:\Users\Didjeeh\Desktop

Name
The name of this item.

Figure 1: vApus v2 Main Window

Before talking about stresstesting and all that it is best to explain vApus v2 at its core, and why it is better than the fundaments in v1, in a nutshell:

- No addin-stuff, SharpDevelop Add-in tree architecture, like it was in the old vApus (Difficult to implement/debug, must do the same stuff per visual studio project, slow compilation)
- Not bloated but light weighted
- No builder classes (saving, loading; showing in the tree view) per vApus solution item but centralized and automated in the framework
- ...

and easily to build new modules because vApus v2 is built like a, as good as, self-managed framework. One quick note, building vApus v2 was first done in Visual Studio 2008 and later in Visual Studio 2010 in C#. .net 4.0 is the used API.

2.1 Static GUI

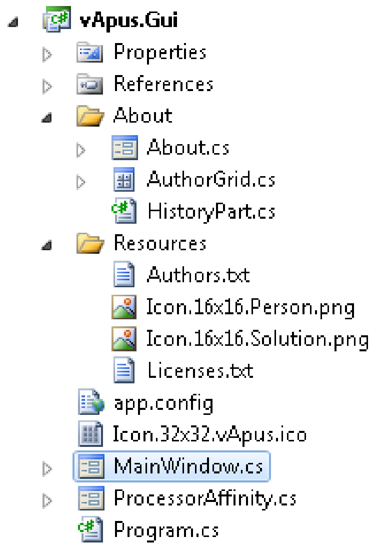


Figure 2: MainWindow.cs in the Visual Studio Solution Explorer

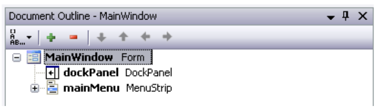


Figure 3: Document Outline of MainWindow.cs

Or the "Hard Coded" GUI. Instead of a standard MDI application there was opted to use a dock panel (http://sourceforge.net/projects/dockpanelsuite) where different forms can be shown as tabs just like in Visual Studio. Note: this was also the case in the previous vApus, only unnecessary classes where implemented upon the referenced .dll. Now the source is a bit adapted and is compiled with vApus v2. (Just a tab page that when trying to close only hides and that can only have one instance.)

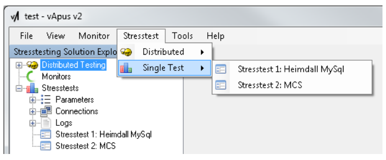
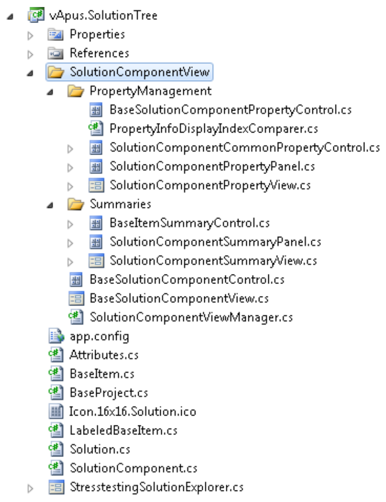


Figure 4: "Hard Coded" GUI

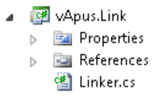
The main menu is static too, for the ease of development. For instance, menu items like "New" and "Open" but also the items "Distributed" and "Single Test" are hard coded. But their childs are not static nor are their images (except "View ? Stresstesting Solution Explorer", which does have a dynamically loaded image). The reason I talk about this first is because there is but a loose connection between the static GUI and the framework, the only thing that deeply interacts is the filling of the menu items "Recent Solutions", "Monitor", "Distributed" and "Single Test". The "About"-folder in the project will be discussed in chapter "Other".

2.2 Managing Modules The Actual Framework



```
vApus.SolutionTree
  Properties
  References
  SolutionComponentView
    PropertyManagement
      BaseSolutionComponentPropertyControl.cs
      PropertyInfoDisplayIndexComparer.cs
      SolutionComponentCommonPropertyControl.cs
      SolutionComponentPropertyPanel.cs
      SolutionComponentPropertyView.cs
    Summaries
      BaseItemSummaryControl.cs
      SolutionComponentSummaryPanel.cs
      SolutionComponentSummaryView.cs
    BaseSolutionComponentControl.cs
    BaseSolutionComponentView.cs
    SolutionComponentViewManager.cs
  app.config
  Attributes.cs
  BaseItem.cs
  BaseProject.cs
  Icon.16x16.Solution.ico
  LabeledBaseItem.cs
  Solution.cs
  SolutionComponent.cs
  StresstestingSolutionExplorer.cs
```

Figure 5: vApus.SolutionTree



```
vApus.Link
  Properties
  References
  Linker.cs
```

Figure 6: vApus.Linker

This is the complete core of ”vApus as a framework”/vApus v2. The six main items are:

- Solution

  This is self-managed, it keeps stuff like:

  – The active solution (can be only one)
  – The recent opened/saved solutions (a list of maximum 10 items)
  – The stresstesting solution explorer
  – The project group types built upon the framework (further)

  and has the functionality to:

  – Register project group types
  – Create new, save , load
  – Build tree nodes (reflection)
  – Notify if the solution has changed

- SolutionComponent

  The base building block for ”BaseProject” and ”BaseItem” (”LabeledBaseItem” inherits from ”BaseItem”) and holds functionality for:

  – Building treenodes and giving back savable data (xml and reflection (attributes))

  - Building a context menu and some basic context menu item invocation targets
  - Getting the appropriate image
  - Adding stuff as standard items
  - Notifies if the component has changed
  - This can be called everywhere and since a static event is invoked, everything will know if something is changed.

```
public void InvokeSolutionComponentChangedEvent(
      SolutionComponentChangedEventArgs.DoneAction doneAction)
public void InvokeSolutionComponentChangedEvent(
      SolutionComponentChangedEventArgs.DoneAction doneAction, object arg)
```

  - ...

  and is approachable like a collection (of "BaseItem"-s).

- Stresstesting Solution Explorer
- BaseSolutionComponentView

  Every view that is shown as dockable content must override from this class. (Further)

- SolutionComponentViewManager

  For every item shown in the solution explorer any view can be shown/managed using this manager (Discussed later on).

- Linker

  Used to register the project groups when the application starts (avoiding circular dependencies).

```
Solution.RegisterProjectType(typeof(StresstestProject));
```

2.3 Generated GUI

2.3.1 Stresstesting Solution Explorer

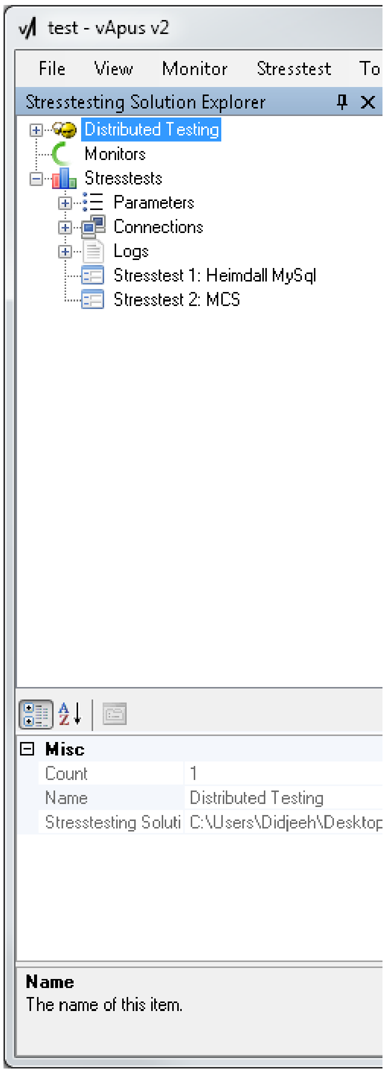
The solution class holds a static solution explorer, which is nothing more than e tree view and property grid. Changes to the solution, like property changes, are invoked with an event.
Solution will notify to the explorer when he should reload completely (new or opened solution) or just some parts (add, remove, clear).
It also provides functionality to edit labels of "LabeledBaseItems".

2.3.2 Tree Nodes

SolutionComponent " gives back its, and its childs, treenodes. It links a context menu to the nodes if applicable.

```
public TreeNode GetTreeNode() {...
```

An image is automatically applied if the resource is provided (see Resource Management).



Count	1
Name	Distributed Testing
Stresstesting Soluti	C:\Users\Didjeeh\Desktop

Name
The name of this item.

Figure 7: Stresstesting Solution Explorer

2.3.3 Context Menu's

Context menu items are generated automatically based on attribute usage and reflection.
If the class attribute "ContextMenu" is provided in a class inheriting from "SolutionComponent" a context menu can be build.

```
[ContextMenu(new string[] { "Add_Click", "SortItemsByLabel_Click", "Clear_Click" }, new
    string[] { "Add Stresstest...", "Sort", "Clear" })]
public class StresstestProject : BaseProject {...
```

The invocation target must be provided (void(object sender, EventArgs e) target), a label is not obliged.
Some are already provided in "SolutionComponent":

-
  ```
  internal void Activate_Click(object sender, EventArgs e){...
  ```

  A virtual function "Activate()" is provided, this is freely to override and do stuff in. By default a "SolutionComponentPropertyView" is shown for the calling type. (Further)

-
  ```
  internal void Clear_Click(object sender, EventArgs e){...
  ```

  Asks for confirmation (message box) and clears.

-
  ```
  internal void SortItemsByLabel_Click(object sender, EventArgs e){...
  ```

  Sorts only the "LabeledBaseItems", the 'normal' ones are put at the beginning.

in "BaseItem":

-
  ```
  internal void Import_Click(object sender, EventArgs e) {...
  \end {codelisting}
  \npar
  Shows an Open File Dialog, asking for a valid xml file. This will check if a
      certain item is valid to be imported.
  \item
  \begin{codelisting}
  internal void Summary_Click(object sender, EventArgs e) {...
  ```

  Discussed later on.

-
  ```
  internal void Remove_Click(object sender, EventArgs e) {...
  ```

  Asks for confirmation (message box) and removes.

-
  ```
  internal void Cut_Click(object sender, EventArgs e) {...
  ```

  ```
  internal void Copy_Click(object sender, EventArgs e) {...
  ```

-
  ```
  internal void Paste_Click(object sender, EventArgs e) {...
  ```

And in "LabeledBaseItem":

-
  ```
  internal void Export_Click(object sender, EventArgs e) {...
  ```

  The label is used as filename.

Hotkeys Hotkeys for the tree nodes can be set the same way as for setting context menu's. Just the class attribute "Hotkeys" must be provided in a class inheriting from "SolutionComponent".

```
[ContextMenu(new string[] { "Edit_Click", "Remove_Click" }, new string[] { "Edit", "
    Remove" })]
[Hotkeys(new string[] { "Edit_Click", "Remove_Click" }, new Keys[] { Keys.Enter, Keys.
    Delete })]
public class StresstestProject : LabeledBaseItem {...
```

Renaming and Activating a Tree Node Tree nodes can be renamed using F2. If a hotkey for "Activate_Click" is provided, double clicking will work too besides the hotkey.

Views For every item shown in the solution explorer any view can be shown/managed using the "SolutionComponentViewManager". This will update and close the views when needed. Views can be made deriving from "BaseSolutionComponentView" and can be shown this way:

```
SolutionComponentViewManager.Show(this, typeof(SolutionComponentPropertyView));
```

Or if the name of the view equals the ToString() of the solution component + "View":

```
SolutionComponentViewManager.Show(this);
```

A default view: "SolutionComponentPropertyView" is already provided holding a "SolutionComponentPropertyPanel" which will add a "SolutionComponentCommonPropertyControl" for each property having the "PropertyControlAttribute". A display index can be given with, if not property controls are alphabetically sorted. ("SolutionComponentPropertyPanel" can also be used freely in other views.)

```
[SavableCloneable, PropertyControl(1)]
public int[] Concurrencies
{
    get { return _concurrencies; }
    set { _concurrencies = value; }
}
```

"SolutionComponentCommonPropertyControl" is applicable for all primary data types and for arrays/generic lists having a primary data type as element type and any object having an IEnumerable as parent (SetParent, GetParent extension method for object (further)). For other types a custom property control can also be provided in the attribute, deriving from "BaseSolutionComponentPropertyControl". When doing this, always override the "Refresh()" function, because this is used to update the control when changes in the properties are made.
The "BaseItem" class also holds a function to show summary controls of each containing item.

```
internal virtual void Summary_Click(object sender, EventArgs e)
{
    SolutionComponentViewManager.Show(this, typeof(SolutionComponentSummaryView));
}
```

As mentioned before, a virtual "Activate()" function has been provided in the "SolutionComponent" class, this is called when clicking the available link label displayed on a summary control. You can override this yourself to show a custom view, by default the "SolutionComponentPropertyView" is shown.

2.4 Resource Management (Images)

Each added module MUST have a "Resources.resx", even if it is always empty. Images for the different items are stored in there having the same name. Using reflection the correct image is found ("SolutionComponent" functionality) which can be (and is) used in menu's and tree views.

```
public Image GetImage(){...
```

2.5 File Management (Automated Saving, Loading and Cloning

```
[SavableCloneable, PropertyControl]
public string Label
{
get { return _label; }
      set { _label = value == null ? string.Empty : value; }
}
```

Using reflection and a property attribute stuff that must be saved can be specified. This way a "BaseProject" and a "BaseItem" can load from and give back xml. This is packaged in "Solution" using the OpenXML way (a zip-file holding xml files). Only primary data types and arrays/lists having a primary data type as element type and any object having an IEnumerable as parent (SetParent, GetParent extension method for object (further)) can be saved and loaded. Note: Properties with the "SavableCloneable" attribute must have a getter and setter and may never be null! Use string.Empty as field value for strings, use BaseItem.Empty for BaseItems. For 'IEnumerable items' a branched index is stored when saving to xml. It starts with the zero-based index of a upper parent and a '.' to branch until the direct parent of the item is reached followed with the index of the item itself. If the object is not a base item the parent field is stored also. Very Important: The parent of the object MUST always be set! Example:

```
_log = BaseItem.Empty(typeof(Log), Solution.ActiveSolution.GetSolutionComponent(typeof(
     Logs))) as Log;
```

Results in:

```
<Log args="vApus.BranchedIndexType">2.2.1</Log>
```

(Stresstests/Logs/Log1)

Furthermore any base items can be cut, copied, pasted, exported and imported. Note: The culture of the main thread is set to "en-US" to make sure there is no issue with floating point numbers (points and comma's mix-up).

2.6 Important Other Stuff

The update and commit logic is adopted from vApus v1 and is basically the same. Also the about dialog is adopted, but now it gets authors and licenses from the resources and the history of changes from the "versioncontrol.ini" if any.

```
<Authors>
<Author name="Dieter Vandroemme" email="dieter@sizingservers.be, dieter.vandroemme@gmail
     .com" period="2006-2010" />
</Authors>
```

Some things here are adopted from the previous vApus, such as "ExtendedListView", "InputDialog" and "StringUtil" who will not be discussed. Stuff that is applicable for stresstesting/monitoring/distributed testing will not be discussed here either.

2.6.1 Comparers and Extensions

These files hold following classes:

- PropertyInfoComparer

  Designed using the singleton design pattern serves at sorting property info by name.

- ControlComparer

  Compares by text, to string, name, full name or a combination of these.

- AssemblyExtension

  A static class which holds an extension method, "GetTypeByName", for Assembly.

- TimeSpanExtension

  Provides functionality to return usable formatted string. Example: 13 minutes, 0 seconds.

- StringExtension

  To encrypt, decrypt, check if valid for Windows filenames and convert, check if is numeric.

- CharExtension

  Check if valid for Windows filenames, is digit and is numeric.

- ObjectExtension

  Set a tag and/or parent. This is a static class with static fields, so an own stack is made using a "HashTable" (very nifty I think). For tag (ditto for parent, wich is used defaultly in the framework when callin Add(BaseItem) on SolutionComponent for example):

```
private static Hashtable _tags = new Hashtable(), _parents = new Hashtable();
public static void SetTag(this object o, object tag)
{
    lock (_tags.SyncRoot)
        if (!_tags.Contains(o))
            _tags.Add(o, tag);
        else
            _tags[o] = tag;
}
public static object GetTag(this object o)
{
    //Threadsafe for reader threads.
    return _tags.Contains(o) ? _tags[o] : null;
}
```

2.6.2 SpecialFolder

This can return all special folders, like "My Documents" or "Desktop".

2.6.3 ClipboardWrapper

Making setting and getting objects to and from the clipboard error safer due to build in retries and error handling.

2.6.4 SocketWrapper

A wrapper around the .net socket class handling synchronous communication and serializes/deserializes objects for you.

2.6.5 SynchronizedContextWrapper

To synchronize with the main thread (GUI) the synchronization context wrapper can be used (the most efficient way of thread synchronization IMHO). Example:

```
SynchronizationContextWrapper.SynchronizationContext.Send(delegate
{ ... }, null);
```

2.6.6 Win32WindowMessageHandler

Serves to register a window message to the application which can be captured in a form using something like this:

```
protected override void WndProc(ref Message m)
{
  if (_msgHandler != null && m.Msg == _msgHandler.WINDOW_MSG)
  {
        this.TopMost = true;
        this.TopMost = false;
        this.Activate();
  }
  base.WndProc(ref m);
}
```

3 Stresstesting with vApus v2 (DR)

3.1 What is Stresstesting?

3.2 A New Approach in vApus v2: Standardized Stresstesting Model

3.2.1 Lexing and Parsing: Rule Sets

3.3 Lexing and Parsing in vApus v2 (D)

3.3.1 Lexing

BaseRuleSet

SyntaxItem

Rule

3.3.2 Lexical Result

ASTNode

3.3.3 Parsing

CustomListParameter

DoubleParameter

StringParameter

3.4 The GUI for All This (and a Quick Note for Researchers)

3.4.1 Parameters

3.4.2 Connections

Connection Proxies (D)

3.4.3 Logs

3.5 Stresstesting

3.5.1 Configuring a Stresstest (and Some Best Practices)

ASAP

CR

3.5.2 Starting a Stresstest and Handling Errors and Other Problems

3.5.3 Saving Results

Cascaded Results

Listed Results

4 Stresstesting with vApus v2: Explaining the Code (D)

4.1 Rule Sets

4.1.1 Lexing

4.1.2 Parsing

4.2 Parameters

4.3 Logs

4.4 Connections

4.4.1 IConnectionProxy and Programming Your Own Connection Proxy Class

4.4.2 ConnectionProxyPool

4.4.3 Free Coding

4.4.4 Unit Testing

Try Compile

Test Code

4.5 Threading

4.5.1 StresstestThreadPool

4.6 StresstestResults

4.7 Stresstesting

4.7.1 Starting a Test

Determining Runs

Determining Test- and Delay Patterns

4.7.2 Executing the Work

4.7.3 Displaying Results

Metric Controls

LogEntryResultsDrawBox

4.7.4 Handling Errors and Other Problems

4.7.5 Stopping a Test

5 Distributed Testing (DR)

5.1 Setting Up a Test (and Some Best Practices)

5.1.1 Resource Pool

5.1.2 Tiles

5.1.3 vApus Jump Start (Experimental)

5.2 Starting a Test, Handling Errors and Handling Other Problems

6 Distributed Testing: Explaining the Code (D)