Browser-based Software for Technology Transfer

Judith Bishop
Jonathan de Halleux
Nikolai Tillmann
Microsoft Research
Redmond, WA, USA
jbishop@microsoft.com

Nigel Horspool
Department of Computer Science
University of Victoria, BC, Canada
nigelh@cs.uvic.ca

Don Syme
Microsoft Research
Cambridge, UK
dsyme@microsoft.com

Tao Xie
Computer Science Department
North Carolina State University, USA
xie@csc.ncsu.edu

ABSTRACT

Technology transfer is typically viewed as being from academia to industry but it can indeed go in either direction. Many of the same challenges then apply – platform suitability, timeliness, support, and community building. In this paper, we describe recent efforts to transfer technology for research and teaching from an industrial research laboratory to universities, and discuss some of the key success factors and major challenges. Examples quoted include Try F# and Pex4Fun.

Categories and Subject Descriptors

D.2 Software Engineering, K.6 Management of Computing and Information Systems

General Terms

Human Factors, Languages

Keywords

Technology transfer, browser-based software, F#, Pex4Fun

INTRODUCTION

Technology transfer is most often viewed as being from a research laboratory to a product group, or from a university research project to industry. Recently, we have been involved in instances of another kind of transfer, from a research laboratory to universities. There are many reasons why this kind of transfer takes place. Sometimes, academics want to make use of a research tool reported in papers. Then they can usually find the tool, download it, and work on it independently.

On the other hand, industrial researchers are increasingly finding that in order to collaborate with university researchers, both need to use the same fairly complex underlying tools. Transferring those tools sometimes means transferring whole platforms as well. Furthermore, there is the issue of training the students who will work on the project. Clearly, it would be better if the students had prior exposure to the technology. With the wide variety of courses taught in universities, this requirement can only be met for very fundamental technologies.

Adoption of tools and technologies widely perceived to be community-based and supported on many platforms, such as Eclipse, Java and Linux, has been easy in principle for universities, but at the start of their lives the tools have required considerable support. Specifically, we have noticed that languages and tools built on the Common Language Infrastructure (CLI) platform face resistance in universities, despite the widespread availability of both professional and open-source implementations of these languages, the respect that the language and framework designs themselves have from the academic community, and their very widespread uptake in industry. Without delving into the reasons or nature of this resistance, this paper looks at recent attempts to overcome it, and endeavours to draw general conclusions about how software can be adapted to lower the barriers and make it more accessible to a wider audience.

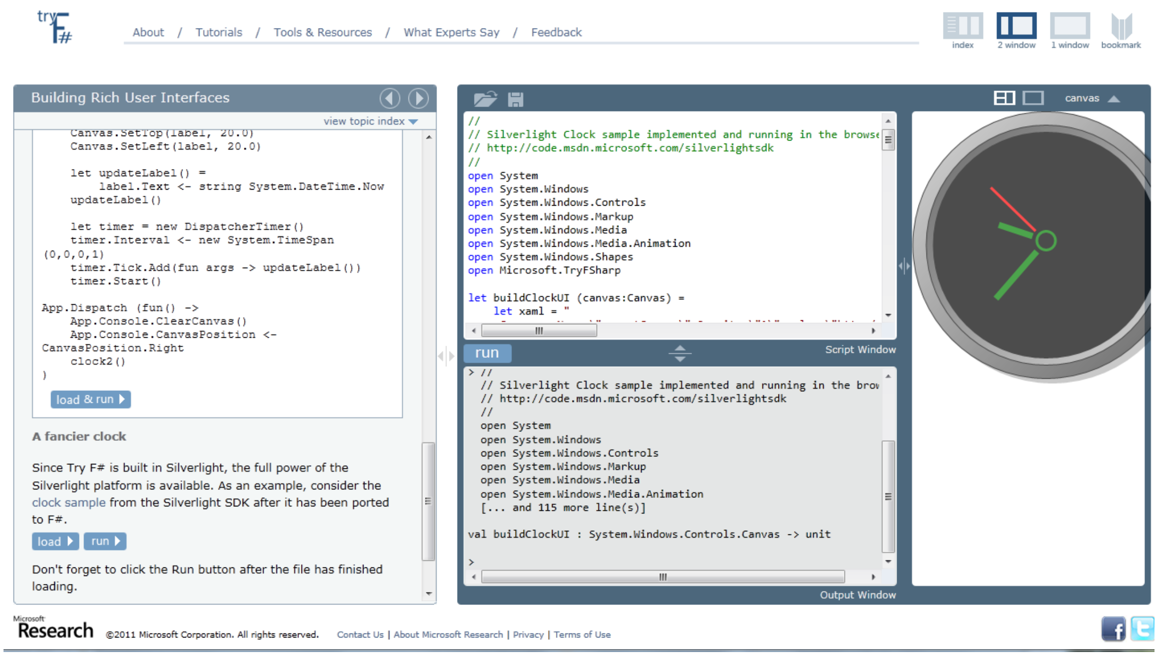
TYPES OF SOFTWARE

The software for language implementations that we are considering for use by universities is grouped depending on whether it requires

1. only a browser, e.g., Explorer, Firefox, Safari
2. a platform and language(s), e.g., a CLI implementation and C# or F#
3. an integrated development environment (IDE), e.g., Visual Studio or Eclipse

From the point of view of broad acceptability, Type 1 language implementations have the advantage of platform independence and simplicity of installation. A disadvantage is that the browser environment is limited in functionality.

For Type 2 language implementations, runtime environments such as .NET, Mono, and the JVM are now commonly part of OS installations or are readily available as add-on packages. Further, Type 2 software is also platform independent, because implementations of the CLI and JVM are available for essentially all modern operating systems. However, the platform dependencies, installation mechanisms, and early learning experience for Type 2 software is still generally more intricate than Type 1 software. If our goal is for students to acquire direct industry-relevant skills, Type 3 language implementations are clearly to be preferred, since the majority of professional programmers use IDE tools as a central part of their work. Further, this situation does not appear to be changing rapidly.


tryF#
About / Tutorials / Tools & Resources / What Experts Say / Feedback
index
2 window
1 window
bookmark
Building Rich User Interfaces
view topic index
load & run
load
run
canvas
run
Script Window
Output Window
Microsoft Research
Contact Us | About Microsoft Research | Privacy | Terms of Use

```
    Canvas.SetTop(label, 20.0)
    Canvas.SetLeft(label, 20.0)

    let updateLabel() =
        label.Text <- string System.DateTime.Now
    updateLabel()

    let timer = new DispatcherTimer()
    timer.Interval <- new System.TimeSpan
(0,0,0,1)
    timer.Tick.Add(fun args -> updateLabel())
    timer.Start()

App.Dispatch (fun() ->
    App.Console.ClearCanvas()
    App.Console.CanvasPosition <-
CanvasPosition.Right
    clock2()
)
```

A fancier clock

Since Try F# is built in Silverlight, the full power of the Silverlight platform is available. As an example, consider the clock sample from the Silverlight SDK after it has been ported to F#.

Don't forget to click the Run button after the file has finished loading.

```
//
// Silverlight Clock sample implemented and running in the browse
// http://code.msdn.microsoft.com/silverlightsdk
//
open System
open System.Windows
open System.Windows.Controls
open System.Windows.Markup
open System.Windows.Media
open System.Windows.Media.Animation
open System.Windows.Shapes
open Microsoft.TryFSharp

let buildClockUI (canvas:Canvas) =
    let xaml = "
```

```
> //
  // Silverlight Clock sample implemented and running in the brow
  // http://code.msdn.microsoft.com/silverlightsdk
  //
  open System
  open System.Windows
  open System.Windows.Controls
  open System.Windows.Markup
  open System.Windows.Media
  open System.Windows.Media.Animation
  [... and 115 more line(s)]

val buildClockUI : System.Windows.Controls.Canvas -> unit

>
```

©2011 Microsoft Corporation. All rights reserved.

Figure 1 Screenshot of TryF#

While Type 3 implementations are usually available for free to universities and people learning languages (e.g., Visual C# Express and MonoDevelop), Type 3 software usually requires additional downloads, and may be, to some extent, tied to particular platforms. Nevertheless, many would argue that the ***ideas*** behind the tools have a wide applicability.

In 2009, researchers in Microsoft Research recognized the limitations of software in types 2 and 3 for disseminating research advances in programming languages to wider communities and started to seek alternatives. We wanted to find a way of presenting the tools without rewriting them or giving up the powerful base of the CLI platform. The result was a concerted effort to make much more use of browser-based software. Other browser-based language software exists, e.g., TryPython and TryRuby, but our twin goals were to use the method seriously for cross-platform compatibility and to introduce some fun into programming.

BROWSER-BASED SOFTWARE

In the past year, we have developed, tested, and deployed two kinds of browser-based software. Two different models for using browsers as the container for the software emerged.

The first is the more sophisticated. A Silverlight/Moonlight control is created to download a complete compiler into a sandbox on the machine. All interaction with the compiler, along with all running of programs, is still done via the browser, any browser. All computation is done on the client.

The second option is to maintain a server, or perhaps a presence on the cloud, and to provide a browser experience that accesses the original system running there.

There are pros and cons to both options. Providing the software to a heterogeneous world of customers requires effort in advance, but no ongoing hardware (apart from the download service). Maintaining a server is easier to set up but has the scalability issue. The server option also has one great advantage: data can be gathered about usage of the software.

Try F#

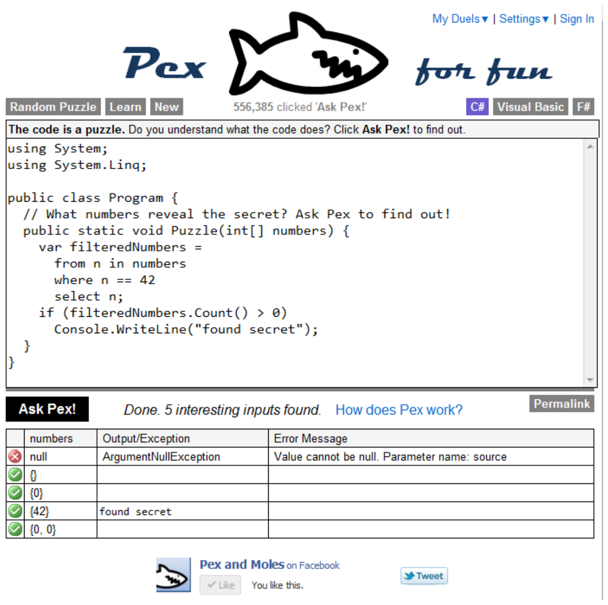
F# [1] is a functional programming language developed by Microsoft Research. Try F# (www.tryfsharp.org) is a tool for running F# in the browser. It was completed in 2011 and is undergoing deployment and testing in the field. When the browser contacts the TryFSharp website, the Silverlight control decides which version of the compiler to download, based on the machine's operating system. The software running in the browser contains not only the compiler, but also a tutorial on the language and a specially written (very important [2]) canvas library for graphical applications. The browser window runs in one, two, or three sections on request. Figure 1 shows a screenshot of TryF#.

Pex4Fun

Pex is a Visual Studio 2010 Power Tool developed by Microsoft Research to help unit testing of .NET applications. It can also be launched from the command line. Thus it can run as Type 2 or Type 3 software. Pex4Fun (www.pex4fun.com) [3,4] is a radically simplified version of the fully featured Pex that is accessed via a browser and all the work happens "in the cloud" (actually on one of Microsoft Research servers). The idea of Pex4Fun is to create a game out of unit testing by providing existing code puzzles in C#, Visual Basic, or F# for users to determine from the unit tests what code needs to be added or changed. Figure 2 shows a screenshot of Pex4Fun.

ASSESSMENT

Browser-based software has proved liberating for tool writers in Microsoft Research. In the Eclipse environment, plug-ins were an easy way of distributing software, but Eclipse is currently far more used in academic research circles as an IDE than Visual Studio is. Now, nearly all the tools from the Research in the Software Engineering (RiSE) Group are available via a portal on www.rise4fun.com. Curious users can try them out, and if they become committed, they can switch to the full version on the full platform.



The code is a puzzle. Do you understand what the code does? Click **Ask Pex!** to find out.

```
using System;
using System.Linq;

public class Program {
  // What numbers reveal the secret? Ask Pex to find out!
  public static void Puzzle(int[] numbers) {
    var filteredNumbers =
      from n in numbers
      where n == 42
      select n;
    if (filteredNumbers.Count() > 0)
      Console.WriteLine("found secret");
  }
}
```

Done. 5 interesting inputs found.

	numbers	Output/Exception	Error Message
✗	null	ArgumentNullException	Value cannot be null. Parameter name: source
✓	{}		
✓	{0}		
✓	{42}	found secret	
✓	{0, 0}		

Figure 2 Screenshot of Pex4Fun

Advertising browser-based software via social media is easier, as is collecting statistics on its use. Thus building a community that takes over the software, which is free or even open-source, is accelerated. In the case of Pex4Fun (which has been running for longer), a typical compliment made online is shown in Figure 3.

Our plans are to further develop both the Try- and –4Fun models further and to investigate how they compare in terms of scalability, maintainability and flexibility to their traditional download versions.

Keith_Flo: This is awesome! I haven't seen anything this cool in a long time! ... People have written 'Cloud' based editors and compilers but this is a whole new level of cool! .. intellisense, multi-languages supported, background compilation .. a debugger is coming .. Wow!

Figure 3. Comment on Pex4Fun

ACKNOWLEDGMENTS

We would like to thank Dean Guo, Dennis Harding, Laurent le Brun, Joe Pamer and Christophe Poulain from Microsoft who wrote and managed Try F#.

REFERENCES

[1] F# 3.0 Reference Manual: http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/spec.pdf

[2] Tim Lammarsch, Wolfgang Aigner, Alessio Bertone, Silvia Miksch, Thomas Turic, Johannes Gaertner, A Comparison of Programming Platforms for Interactive Visualization in Web Browser Based Applications, iv, pp.194-199, 2008 12th International Conference Information Visualisation, 2008

[3] Nikolai Tillmann, Jonathan de Halleux, Tao Xie, Pex for Fun: Engineering an Automated Testing Tool for Serious Games in Computer Science, MSR-TR-2011-41, March 2011

[4] Nikolai Tillmann, Jonathan de Halleux, Tao Xie: Pex4Fun: Teaching and learning computer science via social gaming. CSEE&T 2011: 546-548