PAYING BACK THE TECHNICAL DEBT OF LEGACY APPLICATIONS

don-E Merson
Scholarship Universe
THE SCENARIO

Update this checkbox.
WHAT YOU HAVE TO DO

• You have to add out of date (Archived) scholarships to this dropdown
• You have to order them alphabetically
  • but the active ones should come first
• You have to mark the out of date ones with “(ARCHIVED)”
• So you see the following code
DataTable combinedScholarships = si.GetMyScholarships().Clone();
DataTable dt = si.GetMyScholarships();
for (int i = 0; i < dt.Rows.Count; i++)
{
    if (CheckIfEligibleOnly(i, dt))
    {
        combinedScholarships.ImportRow(dt.Rows[i]);
    }
}
for (int i = 0; i < combinedScholarships.Rows.Count; i++)
{
    combinedScholarships.Rows[i]["Scholarship"] = combinedScholarships.Rows[i]["Scholarship"] + " - Eligible";
}
DataTable dt2 = si.GetAllScholarships();
for (int i = 0; i < dt2.Rows.Count; i++)
{
    for (int j = 0; j < dt.Rows.Count; j++)
    {
        if (dt2.Rows[i]["ScholarshipPackageId"].ToString() == dt.Rows[j]["ScholarshipPackageId"].ToString())
        {
            dt2.Rows.Remove(dt2.Rows[i]);
        }
    }
}
DataTable interestedScholarships = si.GetMyInterestedScholarships();
for (int i = 0; i < interestedScholarships.Rows.Count; i++)
{
    interestedScholarships.Rows[i]["Scholarship"] = interestedScholarships.Rows[i]["Scholarship"] + " - Interested";
}
combinedScholarships.Merge(interestedScholarships);
for (int i = 0; i < appliedScholarships.Rows.Count; i++)
{
    appliedScholarships.Rows[i]["Scholarship"] = appliedScholarships.Rows[i]["Scholarship"] + " - Applied";
}
combinedScholarships.Merge(appliedScholarships);
dataTable awardedScholarships = si.GetMyAwardedScholarships().Tables[0];
for (int i = 0; i < awardedScholarships.Rows.Count; i++)
{
    awardedScholarships.Rows[i]["Scholarship"] = awardedScholarships.Rows[i]["Scholarship"] + " - Awarded";
}
combinedScholarships.Merge(awardedScholarships);
dataTable uninterestedScholarships = si.GetMyTrashedScholarships().Tables[0];
for (int i = 0; i < uninterestedScholarships.Rows.Count; i++)
{
    uninterestedScholarships.Rows[i]["Scholarship"] = uninterestedScholarships.Rows[i]["Scholarship"] + " - Uninterested";
}
combinedScholarships.Merge(uninterestedScholarships);
for (int i = 0; i < dt2.Rows.Count; i++)
{
    dt2.Rows[i]["Scholarship"] = dt2.Rows[i]["Scholarship"] + " - Ineligible";
}
combinedScholarships.Merge(dt2);
for (int i = 0; i < combinedScholarships.Rows.Count; i++)
{
    try
    {
        if (!String.IsNullOrEmpty(combinedScholarships.Rows[i]["DueDate"].ToString()))
        {
            combinedScholarships.Rows[i]["Scholarship"] = combinedScholarships.Rows[i]["Scholarship"] + " - " + combinedScholarships.Rows[i]["DueDate"].ToString().Substring(0, combinedScholarships.Rows[i]["DueDate"].ToString().Length - 15);
        }
        else
        {
            combinedScholarships.Rows[i]["Scholarship"] = combinedScholarships.Rows[i]["Scholarship"] + " - NA";
        }
    }
}
scholarshipSearch.DataTextField = "Scholarship";
scholarshipSearch.DataValueField = "ScholarshipPackageId";
scholarshipSearch.DataSource = combinedScholarships;
scholarshipSearch.DataBind();
YEA IT WAS THAT LONG …

• 82 lines without spacing to be exact
var searchItems =
    DashboardLogic.
    .ScholarshipDashboard.
    .TotalDashboard.
    .Select(
        scholarshipsData =>
        {
            id = scholarshipsData.ScholarshipPackageId,
            text = scholarshipsData.ScholarshipPackage.Scholarship.ScholarshipName + " " +
                + GetScholarshipDateAndDescription (scholarshipsData.ScholarshipPackage.DeadlineDate,
                scholarshipsData.StudentInterestType.StudentInterestDescription),
            orderActiveScholarshipsFirst =
                SetOrderSoActiveScholarshipsAppearFirst (scholarshipsData.ScholarshipPackage.DeadlineDate)
        }
    )
    .OrderBy(s=>s.orderActiveScholarshipsFirst)
    .ThenBy(s=>s.text);

scholarshipSearchDropdownControl.DataSource = searchItems;
scholarshipSearchDropdownControl.DataValueField = "ScholarshipPackageId";
scholarshipSearchDropdownControl.DataTextField = "text";
scholarshipSearchDropdownControl.DataBind();
WHICH WOULD YOU RATHER UPDATE?
WHICH DO YOU THINK WOULD TAKE LONGER TO CHANGE?
WHICH ONE WOULD COST LESS TO CHANGE?
**THIS IS A REAL LIFE EXAMPLE**

- It cost a lot of money
- Programmers ain’t cheap
  - Even the student who coded it
- It also cost a lot of resources
  - First one hits the database 6 times
- It won’t cost as much money now to change
- This is paying back the technical debt
THEN IT CHANGED AGAIN
WHAT IS TECHNICAL DEBT?

• Term coined by Ward Cunningham
• Cost of poor design
• Lack full understanding
SOFTWARE WILL CHANGE

• This bad design costs you money in the future
• Just like a credit card
• Ship it now…
  • Pay for it later
THE THING ABOUT LEGACY APPS

• They always come with debt
• I am going to talk about how to pay off the debt
FIRST PRINCIPLE – WRITE IT FOR READING

Write

Read

1 Time

Read
Read
Read

9 Times
A FUNCTION SHOULD

- Do what it says
- Do One Thing
- Should be totally obvious
- Says WHAT it is doing
- How it is doing it
function bar(number) {
    var nu = 4;
    if (number >= 100) {
        nu++;
        if (window.location.toString().indexOf("ashboard") > 0) {
            //document.getElementById("titleDisp").innerHTML = "<div style="color: red;" class='desc'>" + number + "$</div>"
            //document.getElementById("titleDisp").innerHTML = "<div class='desc'>" + number + "$</div>"
        }
    }
    if (number >= 0) {
        $('#powerbar').html("<div align='center'>" + number + "%</div>");
        //var isMoving = true;
        if (number >= 0) {
            colorPercent();
            $('#powerbar').animate({ height: ((.01 * number) * 303 ) }, 1000);
            //var lastNum = number;
        }
        setTimeout("isMoving = false", 1200);
    }
}
function bar(number) {
  var nu = 4;
  if (number >= 100) {
    nu++;
    if (window.location.toString().indexOf("board") > 0) {
      //document.getElementById("titleDisp").innerHTML = number;
      //document.getElementById("titleDisp").innerHTML = number;
    }
  }
}

if (number >= 0) {
  $('#powerbar').html("<div align='center'>" + number + "</div>");
  var isMoving = true;
  if (number >= 0) {
    colorPercent();
    $('#powerbar').animate({ height: 100 }, 1000);
    //var lastNum = number;
  }
  setTimeout("isMoving = false", 1200);
}
NOW WE KNOW WHY IT IS THERE

```javascript
function renderPowerBarWithAnimation(valueToRender) {
    if (valueToRender >= 0) {
        $('#powerbar').html('<div align="center">" + valueToRender + ""</div>');</n
        colorPercent();

        $('#powerbar').animate({ height: ((.01 * valueToRender) * 303), 1000);
    }

    setTimeout("isMoving = false", 1200);
}
```

STILL WTF
public Calculation GetCalculation
{
    get
    {
        {
            var calculation = StudentInformation.RunCalculationForStudent();
        }
    }
}
public Calculation GetCalculation
{
    get
    {
        if (!StudentCachedCalculationIsNotPresent)
        {
            SetCacheStudentCalculation(StudentInformation.RunCalculationForStudent());
        }
        return CachedStudentCalculation;
    }
}

private Calculation CachedStudentCalculation
{
    get
    {
    }
}

private void SetCacheStudentCalculation(Calculation calculation)
{
}

private static bool StudentCachedCalculationIsNotPresent
{
    get
    {
    }
}
• Actually does 2 things
• Gets
• Sets if not present
• Uses a negative
• Be clear, not pure
A VARIABLE SHOULD TELL YOU
//Specify bgcolor of calendar
var bg = "#5C5C5D";

//Specify font size
var fs = 2;
var flg = 0;
var fs = 1;
var D = new Array("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat");
var calendarString = "";
var monthForwardValue = 0;
var monthBackwardValue = 0;
var mo, yr, bgn;
BETTER-TELLS WHY THEY ARE THERE

```javascript
var backgroundColor = "#5C5C5D";
var fontSize = 2;
var flagForSwitch = 0;
var stringToHoldCalendarHTML = "";
var monthForwardValue = 0;
var monthBackwardValue = 0;
var currentMonth, currentYear, dateToRender;
```

```javascript
//Specify bgcolor of calendar
var bg = "#5C5C5D";
//Specify font size
var fs = 2;
var flg = 0;
var fs = 1;
var D = new Array("Sun", "Mon", "Tues", "Wed", "Thur", "Fri", "Sat");
var calendarString = "";
var monthForwardValue = 0;
var monthBackwardValue = 0;
var mo, yr, bgn;
```
WHY ARE THESE IN THE PROGRAM?

```
public List<StudentApplicationStatus> EligibleScholarships {get; set; }
public List<StudentApplicationStatus> ArchivedScholarships {get; set; }
public List<StudentApplicationStatus> InterestedScholarships {get; set; }
public List<StudentApplicationStatus> AppliedScholarships {get; set; }
public List<StudentApplicationStatus> TrashedScholarships {get; set; }
public List<StudentApplicationStatus> AwardedScholarships {get; set; }
public List<StudentApplicationStatus> NonQualifiedScholarships {get; set; }
public List<StudentApplicationStatus> UnknownScholarships {get; set; }
public List<StudentApplicationStatus> AllScholarships { get; set; }
```
2ND PRINCIPLES-NO MAGIC VALUES

```csharp
public List<StudentApplicationStatus> InterestedScholarships
{
    get
    {
        return AllScholarships.Where(s => s.ScholarshipPackage.DeadlineDate >= DateToUse &&
                                          s.StudentInterestTypeId == SUVariables.StudentScholarshipAction.Interested)
                                .ToList();
    }
}
```
public List<StudentApplicationStatus> EligibleScholarships
{
   get
   {
      return
      AllScholarships
                      .Where(s => scholarship.ScholarshipPackage.DeadlineDate >= DateToUse)
                      .Where(s => s.StudentInterestTypeId == SUVariables.DashboardActionIds.Qualified
                                  || s.StudentInterestTypeId == SUVariables.StudentScholarshipAction.Viewed)
                      .ToList(); //this doesn't work as one long query, DONT combine these where clauses
SHOULD BE ABLE TO PASS THE KEN TEST

• An intelligent person who understands the application should be able to understand the top level of your code
• Ken is my manager so I call it the Ken test
• But more importantly…
THESE LEAD TO TESTABLE CODE

• You can test
• IsStudentCacheCalculationPresent
• GetStudentCalculation
• EligibleScholarships
• How does the Bi-Weekly Campaign work?
BUT LEGACY CODE USUALLY HAS ONE LARGE PROBLEM
CALLS TO THE DATABASE

• Inside Functions
• Inside Constructor
• Inline everywhere
• Can’t test that
4th Principle - Pass Data

- Don’t call the database within
  - Constructor
  - Function
- Always pass it
- Or a Repository
```csharp
public Calculation(List<StudentAttribute> studentAttributes,
                   List<GetScholarshipWizardData_Result> scholarshipRequirements, DateTime? dateToUse = null)
{
    if (StudentAttributesAreAbsent(studentAttributes))
    {
        return; // can't do anything
    }
    CurrentDateTime = dateToUse == null ? DateTime.Now : DateTime.Parse(dateToUse.ToString());
    InitialScholarshipResultsSent = scholarshipRequirements;
    if (StudentAttributesAreAbsent(studentAttributes))
    {
        return; // can't do anything
    }
    StudentData = studentAttributes;
    ScholarshipRequirements = scholarshipRequirements;
    StudentRequirementCalculations = new List<StudentRequirementCalculation>()
    { CurrentEMPLID = StudentData.First().EMPLID;
      ScholarshipPackageIds = scholarshipRequirements.Select(s => s.ScholarshipPackageId).Distinct().ToList();
      InitializeTheScholarshipLogicGroupList(scholarshipRequirements);
      CalculateEachItem();
    }
}
```
public ScholarshipDashboardLogic(StudentInformation studentInformation, ScholarshipUniverseEntities database) {
    this.StudentInformation = studentInformation;
    this.Database = database;
}
5TH PRINCIPLE - WRITE TESTS

• Unit tests
• Integration tests
• Regression tests
• These allow you to do the first 4 principles
• Without fear
[TestMethod]
public void VerifyCalculationClassCanTakeSPs()
{
    var studentAttributes = new List<StudentAttribute>();
    var scholarships = new List<GetScholarshipWizardData_Result>();
    var currentAttributes = new List<Attribute>();
    using (var db = new ScholarshipUniverseEntities())
    {
        studentAttributes = db.StudentAttributes.Where(s => s.EMPLID == "22050080").ToList();
        scholarships = db.GetScholarshipWizardData(DateTime.Parse("08/21/2014")).ToList();
        currentAttributes = db.Attributes.ToList();
    }
    var calculation = new Calculation(studentAttributes, scholarships);
    Assert.AreEqual(studentAttributes.Count, calculation.StudentData.Count);
    Assert.AreEqual(scholarships.Count, calculation.ScholarshipRequirements.Count);
    Assert.AreNotEqual(0, calculation.StudentRequirementCalculations.Count);
}
public void VerifyQualifiedScholarshipWithOneAttributeTest()
{
    var studentAttributes = new List<StudentAttribute>();
    var studentAttribute1 = new StudentAttribute();
    studentAttribute1.AttributeId = 229;
    studentAttribute1.EMPLID = "22050080";
    studentAttribute1.AttributeValue = ",Test1,";
    studentAttributes.Add(studentAttribute1);
    var scholarshipsList = new List<GetScholarshipWizardData_Result>();
    var scholarshipReq = new GetScholarshipWizardData_Result();
    scholarshipReq.AttributeId = 229;
    scholarshipReq.RequirementTypeCode = "*";
    scholarshipReq.RequirementValue = ",Test1,;"
    scholarshipReq.AttributeTypeId = 4;
    scholarshipsList.Add(scholarshipReq);
    var calc = new Calculation(studentAttributes, scholarshipsList);
    Assert.IsNotNull(calc);
    Assert.AreEqual(1, calc.QualifiedScholarships.Count());
}
THE RESULT

• Implement the boy scout code

  Leave the code better than you found it
[TestMethod]
public void VerifyCalculationClassCanTakeSPs()
{
    var studentAttributes = new List<StudentAttribute>();
    var scholarships = new List<GetScholarshipWizardData_Result>();
    var currentAttributes = new List<Attribute>();
    using (var db = new ScholarshipUniverseEntities())
    {
        studentAttributes = db.StudentAttributes.Where(s => s.EMPLID == "22050080").ToList();
        scholarships = db.GetScholarshipWizardData(DateTime.Parse("08/21/2014")).ToList();
        currentAttributes = db.Attributes.ToList();
    }
    var calculation = new Calculation(studentAttributes, scholarships);
    Assert.AreEqual(studentAttributes.Count, calculation.StudentData.Count);
    Assert.AreEqual(scholarships.Count, calculation.ScholarshipRequirements.Count);
    Assert.AreEqual(0, calculation.StudentRequirementCalculations.Count);
}
[TestMethod]
public void VerifyCalculationClassCanTakeSPs()
{
    var studentAttributes = new List<StudentAttribute>();
    var scholarships = new List<GetScholarshipWizardData_Result>();

    using (var db = new ScholarshipUniverseEntities())
    {
        studentAttributes = db.StudentAttributes.Where(s => s.EMPLID == "22050080").ToList();
        scholarships = db.GetScholarshipWizardData(DateTime.Parse("08/21/2014")).ToList();
    }

    var calculation = new Calculation(studentAttributes, scholarships);
    Assert.AreEqual(
        studentAttributes.Count, calculation.StudentData.Count);
    Assert.AreEqual(scholarships.Count, calculation.ScholarshipRequirements.Count);
    Assert.AreEqual(calculation.ScholarshipRequirements.Count, calculation.StudentRequirementCalculations.Count);
}
IN CONCLUSION

• Legacy code usually passes on debt to you
• You can repay the debt and stop the madness
• Use 5 principles
  • Write it for reading
  • Remove Magic Values
  • Only comment if necessary
  • Pass Data
  • Write Tests
• It will take you less time in the long run
AND THIS WILL CAN BE YOU