Seven Diagrams Every Software Developer Should Understand
Also Known As

“How Not to be Surprised in Software Development”
These materials are © 1996-2014 Construx Software Builders, Inc.

All Rights Reserved. No part of the contents of this seminar may be reproduced or transmitted in any form or by any means without the written permission of Construx Software Builders, Inc.
class Class1 {

  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  /// <param name="args">Main(string[] args)

  // Logon
  LicenseManager licenseManager = new LicenseManager();

  string networkLicense = "C:\Documents and Settings\user.XPATH\My Documents\My Videos\South Park-(Freak Strike)-2004-08-17-0.mpg";

  licenseManager.LogOn(networkLicense, password);

  Console.WriteLine("Logged on.");

  string FullName = @"C:\Documents and Settings\user.XPATH\My Documents\My Videos\South Park-(Freak Strike)-2004-08-17-0.mpg";

  BTULibrary library = new BTULibrary();

  // Get properties
  FUPPropertyBag bag = library.GetMediaByFullName(FullName);

  // Print properties to the console
  Console.WriteLine("Properties of ", FullName);
  foreach(FUPProperty prop in bag.Properties) {
    Console.WriteLine("Property: ", (int) prop.Name, prop.Value);
  }

  // Put the FUPPropertyBag into a more friendly collection class.
  // It's a good idea for you to write a friendlier wrapper class that
  // would allow you to add and remove properties and cast back to
  // the FUPPropertyBag type on the fly.
  ArrayList properties = new ArrayList(bag.Properties);

  // Change the "EpisodeDescription" property
  foreach(FUPProperty prop in properties) {
    if (prop.Name == "EpisodeDescription") {
      prop.Value = "The boys compete to appear on a talk show. (Edited by Beyond TV Framework)"
    }
  }

  // Create a new FUPPropertyBag with the edited property
  FUPPropertyBag newBag = new FUPPropertyBag();
  newBag.Properties = (FUPProperty[]) properties.ToArray(typeof(FUPProperty));

  // This method will edit the recording
  library.EditMedia(FullName, newBag);

  // Print properties to the console and verify the change
  Console.WriteLine("Edited properties of ", FullName);
  foreach(FUPProperty prop in bag.Properties) {
    Console.WriteLine("Property: ", (int) prop.Name, prop.Value);
  }

  // Pause so you can see the output. hit enter to continue
  Console.WriteLine("Press any key to exit...");
  Console.ReadKey();
  return;
}
How Not to Be Surprised in Software Development

My Background

300 books and articles?

600 books and articles!

Another 300 books and articles!

Another ~1000 papers, of which only 250 were publishable

<sigh>
I did not work as hard on my next book, *Software Project Survival Guide* ...
A History of Attempts to Explain Software Development

class Class1 {
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    /// <param name="args">The command-line arguments.</param>
    static void Main(string[] args) {
        // Import
        BTULicenseManager LicenseManager = new BTULicenseManager();
        string networkLicense = "networklicense";
        string password = "networklicense">
            licenseManager.Logon(networkLicense, password);
        Console.WriteLine("Logged on.");
        string filePath = "C:\Documents and Settings\User\My Documents\My Videos\South Park\From the Ashes-2004-08-17-0.mpg";
        BTUPlayer player = new BTUPlayer();
        string outputName = "output.mpg";
        // Print properties to the console
        Console.WriteLine("Properties of ");
        foreach (PUSTriProperty prop in bag.Properties) {
            Console.WriteLine("Property: ", prop.Name, prop.Value);
        }
        // Create an abstract base class.
        // Note that a class that includes this class as an Interface cast back to
        // Arraylist aProperties = new ArrayList(bag.Properties);
        // Change the "EpisodeDescription" property
        foreach (PUSTriProperty prop in aProperties) {
            if (prop.Name == "EpisodeDescription") {
                prop.Value = "Edited by some TV framework";
            }
        }
        // Print properties to the console and verify the change
        Console.WriteLine("Edited properties of ");
        foreach (PUSTriProperty prop in bag.Properties) {
            Console.WriteLine("Property: ", prop.Name, prop.Value);
        }
        // Pause so you can see the output. Hit enter to continue
        Console.WriteLine("Press any key to exit...");
        Console.ReadLine();
        return;
    }
}
A Long Line of Attempts to Explain Software Development
Why do we Need to Help People Understand Software Engineering?
Why do we Need to Help Other People Understand Software Engineering? (cont.)

During the House hearing, contractors said CMS decided at the last minute not to allow people to shop for plans before learning what kinds of tax credits they might receive.

“… they had just two weeks to test the site before all the pieces from several contractors had to work together the day of the launch.”

“We all know we were working under a compressed time frame to launch this on Oct. 1.”

“... they had just two weeks to test the site before all the pieces from several contractors had to work together the day of the launch.”

“Determining many of the problems the system would have after the various parts were integrated was difficult until the site actually went online, Bataille said. It was the agency’s responsibility to make sure all the parts worked together.

“According to one specialist, the Web site contains about 500 million lines of software code. By comparison, a large bank’s computer system is typically about one-fifth that size.”

“The technology is there to do that. It just requires foresight.”

This is “Not Enough Testing”???
Latest Attempt to Explain Software Development

Software Engineering Essentials Lecture Series

CxLearn.com
The Goal

Help software professionals develop mental models and frameworks to understand and improve their software projects, to understand the context for current software practices, and to evaluate new developments in software engineering.
Talk Roadmap
Introduce four core influences
Highlight seven key diagrams
Introduce many other diagrams that are also informative
What’s the benefit of that?
Four Core Software Influences

```csharp
class Class1 {

    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string[] args) {

        // LicenseManager licenseManager = new BTULicenseManager();
        // licenseManager.logIn( "00000000-0000-0000-0000-000000000000", "00000000-0000-0000-0000-000000000000" , "password", licenseManager.LogIn(networklicense, password));

        Console.WriteLine( "Logged on." );

        // Print properties to the console
        Console.WriteLine("Properties of <02">; Full Name;"
        foreach(PUSProperty prop in bag.Properties) {
            Console.WriteLine("Property: <02», (1)" propName, prop.Value;"
        }

        // Put the PUSPropertyBag into a more friendly collection class.
        // It's a good idea for you to write a friendlier wrapper class that
        // would allow you to add and remove properties and cast back to
        // the PUSPropertyBag type on the fly.
        arraylist<Integer> properties = new arraylist<>( bag.Properties;"

        // Change the "Episode Description" property
        foreach(PUSProperty prop in aProperties) {
            if(prop.Name = "Episode Description") {
                prop.Value = "The boys compete to appear on a talk show. <Edited by Beyond TV Framework>";"
            }
        }

        // Create a new PUSPropertyBag with the edited property
        PUSPropertyBag newBag = new PUSPropertyBag();
        newBag.Properties = new PUSPropertyBag().Zip(aProperties, typeof(PUSProperty));

        // This method will edit the recording
        library.EditMedia( Full Name, newBag;"

        // Print properties to the console and verify the change
        Console.WriteLine("Edited properties of <02" Full Name;"
        foreach(PUSProperty prop in bag.Properties) {
            Console.WriteLine("Property: <02", (1)" prop.Name, prop.Value;"
        }

        // Pause so you can see the output, hit enter to continue
        Console.WriteLine("Press any key to exit...";"
        Console.ReadLine();
        return;
    }
}
```
Four Core Influences

**SIZE** (diseconomy of scale; failure rate; specializations; mix of activities)

**UNCERTAINTY** (intellectual phases; cone of uncertainty; feature staircase vs. feature buildup; risk management; effort vs. certainty curve)

**DEFECTS** (DCI, defect detection lag, defect removal techniques in series, relationship to process stability)

**HUMAN VARIATION** (effect on research; effect on selection of methods (familiar vs. unfamiliar); effect on team composition, team cohesion, recruiting, and retention; focus on perfect execution vs. perfect plans; implication for favoring robust methods)
Core Influence: Size
Which “Size” Diagrams Are Most Informative?

We have many informative diagrams. Which really explains the essence of Size in Software?
Efficiency is an $N^2$ function of the number of people on the project due to communication paths: $N(N-1)/2$
Diseconomy of Scale (Larry Putnam)

Adapted from Lawrence H. Putnam and Ware Myers, *Five Core Metrics: The Intelligence Behind Successful Software Management*
Brooks’ Diseconomy of Scale Revisited

Efficiency is an $N^2$ function of the number of people on the project due to communication paths: $N*(N-1)/2$

1 person = 0 paths
2 people = 1 path
3 people = 3 paths
4 people = 6 paths
5 people = 10 paths
50 people = 1225 paths
Diseconomy of Scale: McConnell’s Step Function

![Graph showing the diseconomy of scale with productivity dropping as team size increases from 1 to approximately 250-350 members.](image-url)
Diseconomy of Scale: McConnell’s Step Function, Output
Failure Rates by Size

Notice the “High” here? This implies that a 500 MLOC healthcare.gov would require at least 100,000 staff-years of effort (all since 2010!)
### Specializations by Organization Size

#### Table 15-2. Appropriate specializations by company size

<table>
<thead>
<tr>
<th>Specialty</th>
<th>Number of Software Employees</th>
<th>Ratio to Generalists</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proportion of Specialists</td>
<td>0%</td>
<td>-</td>
</tr>
<tr>
<td>Architecture</td>
<td>X</td>
<td>1:75</td>
</tr>
<tr>
<td>Configuration Control</td>
<td>X</td>
<td>1:30</td>
</tr>
<tr>
<td>Cost estimating</td>
<td>X</td>
<td>1:100</td>
</tr>
<tr>
<td>Customer support</td>
<td>X</td>
<td>1:25*</td>
</tr>
<tr>
<td>Database administration</td>
<td>X</td>
<td>1:25</td>
</tr>
<tr>
<td>Education and training</td>
<td>X</td>
<td>1:250</td>
</tr>
</tbody>
</table>

Note: X indicates the appropriate range for the specialty based on company size.
Cocomo Factors by Size
Cocomo II’s View of Software Project Influences

- Product Complexity: -27%
- Requirements Analyst Capability: -29%
- Programmer Capability (general): -24%
- Time Constraint: 0%
- Personnel Continuity (turnover): -19%
- Multi-Site Development: -22%
- Required Software Reliability: -18%
- Extent of Documentation Required: -19%
- Applications (Business Area) Experience: -19%
- Use of Software Tools: -22%
- Platform Volatility: -13%
- Storage Constraint: 0%
- Precededness: -19%
- Process Maturity: -19%
- Language and Tools Experience: -16%
- Database Size: -10%
- Platform Experience: -15%
- Architecture and Risk Resolution: -18%
- Development Flexibility: -16%
- Developed for Reuse: -5%
- Team Cohesion: -14%

Influences:
- Product Complexity: 74%
- Requirements Analyst Capability: 42%
- Programmer Capability (general): 34%
- Time Constraint: 63%
- Personnel Continuity (turnover): 29%
- Multi-Site Development: 22%
- Required Software Reliability: 26%
- Extent of Documentation Required: 23%
- Applications (Business Area) Experience: 22%
- Use of Software Tools: 17%
- Platform Volatility: 30%
- Storage Constraint: 46%
- Precededness: 15%
- Process Maturity: 15%
- Language and Tools Experience: 20%
- Database Size: 28%
- Platform Experience: 19%
- Architecture and Risk Resolution: 14%
- Development Flexibility: 12%
- Developed for Reuse: 24%
- Team Cohesion: 11%
Cocomo II’s View of Software Project Influences

- Product Complexity: -27%
- Requirements Analyst Capability: -29%
- Programmer Capability (general): -24%
- Time Constraint: 0%
- Personnel Continuity (turnover): -19%
- Multi-Site Development: -22%
- Required Software Reliability: -18%
- Extent of Documentation Required: -19%
- Applications (Business Area) Experience: -19%
- Use of Software Tools: -22%
- Platform Volatility: -13%
- Storage Constraint: 0%
- Precededness: -19%
- Process Maturity: -19%
- Language and Tools Experience: -16%
- Database Size: -10%
- Platform Experience: -15%
- Architecture and Risk Resolution: -18%
- Development Flexibility: -16%
- Developed for Reuse: -5%
- Team Cohesion: -14%
Cocomo II’s View of Software Project Influences

- Product Complexity: -27% Positive, 74% Negative
- Requirements Analyst Capability: -29% Positive, 42% Negative
- Precedent: -33% Positive, 30% Negative
- Process Maturity: -33% Positive, 30% Negative
- Architecture and Risk Resolution: -30% Positive, 27% Negative
- Programmer Capability (general): -24% Positive, 34% Negative
- Development Flexibility: -27% Positive, 24% Negative
- Time Constraint: 0% Positive, 63% Negative
- Team Cohesion: -24% Positive, 21% Negative
- Personnel Continuity (turnover): -19% Positive, 29% Negative
- Multi-Site Development: -22% Positive, 22% Negative
- Required Software Reliability: -18% Positive, 26% Negative
- Extent of Documentation Required: -19% Positive, 23% Negative
- Applications (Business Area) Experience: -19% Positive, 22% Negative
- Use of Software Tools: -22% Positive, 17% Negative
- Platform Volatility: -13% Positive, 30% Negative
- Storage Constraint: 0% Positive, 46% Negative
- Language and Tools Experience: -16% Positive, 20% Negative
- Database Size: -10% Positive, 28% Negative
- Platform Experience: -15% Positive, 19% Negative
- Developed for Reuse: -5% Positive, 24% Negative
Why Does This Matter?  
(Implications of the Diagram)

- Understanding the factors in the Cocomo model and their relative importances goes a long way toward explaining software project dynamics overall.
- Many dynamics related to **project size** are implied by Cocomo’s scaling factors.
- Many dynamics related to **human variation** are implied in the Cocomo model.
Activity Mix by Project Size
Project Activity Mix by Project Size

Construction is approx. 2/3
Construction is approx. 1/3

- System Test
- Construction
- Architecture
- Requirements
Why Does This Matter? (Implications of the Diagram)

- Small project success does not prepare organizations for large project success
- Organizations must change focus as their projects inevitably become larger
- Organizations must build different/additional skill sets as their projects become larger
- There are numerous interactions between size, uncertainty, defects, and human variation
Core Influence: Uncertainty
Again, We Have Many Wonderful “Uncertainty” Diagrams
Effort vs. Knowledge Curve
Feature Build Down (The Feature Staircase)
Feature Build Up

Diminishing returns when functionality is delivered in priority order

Opportunity to Add More Value

Value / Functionality Cost

Time
Risk Management: Relationship between Planned and Unplanned Work

Planned Work

Planned "Overhead"

Unplanned, Variable Work
Cone of Uncertainty vs. Cloud of Uncertainty
Cone of Uncertainty vs. Cloud of Uncertainty

Project schedule

Project scope (effort, size, features)

Initial product definition
Approved product definition
Detailed design
Product complete

0.25x
0.5x
0.67x
0.8x
1.0x
1.25x
1.5x
2x
4x
0.6x
0.8x
1.0x
1.1x
1.15x
1.25x
1.6x

Official Top 7 Diagram
Why Does This Matter?
(Implications of the Diagram)

- Explains why perfect estimation on Day 1 of a project is not possible
- Explains why reestimation is necessary if you want accurate estimates
- Explains why actively attacking uncertainty is essential
class Class1 {

    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string args) {
        // Logon
        LicenseManager licenseManager = new LicenseManager();
        string networkLicense = "c:\documents and settings\user\snapstream\my documents\my videos\south park-(freak strike)-2004-09-17-0.mpg";
        string password = "";
        licenseManager.Logon(networkLicense, password);
        Console.WriteLine("Logged on.");
        string fullName = networkLicense;
        string[] properties = new string[] { "fullName", "author", "description", "copyright", " Crown Copyright 2009. All rights reserved. " };
       _propertyBag = library.GetMediaByFullName(fullName);
        // Print properties
        Console.WriteLine("Properties of "{0}". ", fullName); 
        foreach (PUSEProperty prop in _propertyBag.Properties) { 
            Console.WriteLine("Property: "{0}, (1)" prop.Name, prop.Value); 
        } 
        // Put the PUSEPropertyBag into a more friendly collection class.
        // It's a good idea for you to write a friendlier wrapper class that
        // would allow you to add and remove properties and cast back to
        // the PUSEPropertyBag type on the fly.
        array<int> _properties = new Array<int>(_propertyBag.Properties);
        // Change the "EpisodeDescription" property
        foreach (PUSEProperty prop in _properties) {
            if (prop.Name == "EpisodeDescription") {
                prop.Value = "The boys compete to appear on a talk show. (Edited by Beyond TV)";
            }
        }
        // Create a new PUSEPropertyBag with the edited property
        PUSEPropertyBag newBag = new PUSEPropertyBag();
        newBag.Properties = Convert.ToObject<Array<int>>(_propertyBag.Properties);
        // This method will edit the recording
        library.EditMedia(fullName, newBag);
        // Print properties to the console and verify the change
        Console.WriteLine("Edited properties of "{0}". ", fullName); 
        foreach (PUSEProperty prop in newBag.Properties) { 
            Console.WriteLine("Property: "{0}, (1)" prop.Name, prop.Value); 
        } 
        // Pause so you can see the output, hit enter to continue
        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
        return;
    }
}
This figure is adapted from Grady Booch, *Object Solutions: Managing the Object-Oriented Project*, Reading, Mass: Addison Wesley 1996.
Intellectual Phases

Cost of Overlap

Overlap =
- Dependencies
- Uncertainty
- Risk
- Rework
- Higher costs
- Longer schedules
- Lower quality
Intellectual Phases
Degree of Overlap

Focus

Discovery Invention Construction

Discovery Invention Construction

Discovery Invention Construction

Time

Construx®
Intellectual Phases
Variations in Sources of Project Challenges

Focus

Discovery  Invention  Construction

Construx®
Why Does This Matter? (implications of the diagram)

- Many project failures are caused by treating uncertainties as though they are certain.
- Additional inefficiencies are created by treating uncertainties as if they were certain.
- Inefficiencies are also created by treating certainties as if they were uncertain.
- Software’s intangibility exacerbates this phenomenon.
- Diagram provides a framework for identifying uncertainty and planning appropriately.
Core Influence:

Defects
Defect Cost Increase (DCI)

Activity in which a Defect Is Introduced

Requirements
Architecture
Construction

Requirements Architecture Construction System test Post-Release

Activity in Which a Defect Is Detected

Average Cost to Correct
Fix More Defects Earlier!

Activity in which a Defect is Introduced

Fix Here

Don’t Wait to Fix Here

Activity in Which a Defect is Detected

Requirements

Architecture

Construction

Average Cost to Correct

Requirements Architecture Construction System test Post-Release

Fix Here

Don’t Wait to Fix Here
Reduce Defect Cost Increase!

Activity in which a Defect Is Introduced

Requirements
Architecture
Construction

Requirements
Architecture
Construction
System test
Post-Release

Activity in Which a Defect Is Detected

Average Cost to Correct
Quality is an Accelerator

Quality improvement motivated primarily by economics (quality is free)

Quality improvement motivated by quality, per se (quality costs more)

Most Org’s are Here

Percentage of Defects Removed Before Release

~95%  100%

Effort/ Cost/ Schedule

54
Gap Between Defect Insertion and Defect Detection
Minimize Gap Between Defect Insertion and Defect Detection

Typical Project

Well-Run Project

Cumulative Defects

Risk of Extra Cost

Defect creation

Defect removal

Time

Risk of Extra Cost

Schedule / Cost Savings

Defect creation

Defect removal

Time

Typical Project

Well-Run Project

Cumulative Defects

Risk of Extra Cost

Defect creation

Defect removal

Time

Risk of Extra Cost

Schedule / Cost Savings

Defect creation

Defect removal

Time

Risk of Extra Cost

Schedule / Cost Savings

Defect creation

Defect removal

Time
Why Does This Matter?

- Practices that increase gaps between defect insertion and defect detection will increase project costs and project risk.
- Practices that minimize gaps between defect insertion and defect detection will reduce project costs and project risks.
- Understanding this diagram fully will lead to basically same conclusions as understanding the other diagrams.
Core Influence:

Human Variation
This is not just general “humans develop software.” This is specifically about variation across different humans.

**Human Beings Exhibit the Same Variations in Performance That Programmers Do!**

Of the four areas of rapid-development leverage—people, process, product, and technology—“people” has the greatest potential to shorten software schedules across a variety of projects. Most people who work in the software industry have personally observed the enormous differences in output between average developers, mediocre developers, and genius developers. Researchers have identified performance differences on the order of 10 to 1 or more between different developers with the same levels of experience (Sackman, Erikson, and Grant 1968, Curtis 1981, Mills 1983, DeMarco and Lister 1985, Curtis et al. 1986, Card 1987, Valett and McGarry 1989).

Motivation is undoubtedly the key to tapping into motivation to improve development speed. Although motivation is a soft factor, the knowledge of how to motivate software developers is not a total mystery. This chapter describes how to tap into motivation to improve development speed.

Of the four areas of rapid-development leverage—people, process, product, and technology—“people” has the greatest potential to shorten software schedules across a variety of projects. Most people who work in the software industry have personally observed the enormous differences in output between average developers, mediocre developers, and genius developers. Researchers have identified performance differences on the order of 10 to 1 or more between different developers with the same levels of experience (Sackman, Erikson, and Grant 1968, Curtis 1981, Mills 1983, DeMarco and Lister 1985, Curtis et al. 1986, Card 1987, Valett and McGarry 1989).
Cocomo II’s View of Software Project Influences

- Requirements Analyst Capability: -29%, 42%
- Programmer Capability (general): -24%, 34%
- Personnel Continuity (turnover): -19%, 29%
- Applications (Business Area) Experience: -19%, 22%
- Language and Tools Experience: -16%, 20%
- Platform Experience: -15%, 19%
- Team Cohesion: -14%, 11%

Cumulative Effect of Personnel Factors: -78%, 373%
Human Variation vs. Process-or-Practice Variation
Differences in Productivity

Productivity

A  Productivity of Team A
B  Productivity of Team B
Differences in Methods

Productivity

- Team A Used Pair Programming
- Team B Used Formal Inspections

Which method is better?
Differences in Capability

Productivity

A  Team A Had Star Performers
B  Team B Had Average Performers

Now which method is better?
Process Variation vs. Human Variation

Typical Variation in Individual Productivity (20:1) and Team Productivity (3-10:1)

Typical Variation in Method Productivity (~20%)
Effect of Variations in Human Productivity

Is the observed productivity difference between A and B due to method differences or to differences in individual capability or team capability?
Example: Chrysler C3 Extreme Programming Project

Range that Kent Beck and Ron Jeffries would perform using any methods whatsoever

Kent and Ron’s performance on Chrysler C3 project (speculation)

Range that average personnel will perform in, with or without Extreme Programming

So, what is the effect of Extreme Programming?
Why Does This Matter?

- Academic research on process effectiveness must account for human variation to be meaningful (and most does not)
- Evaluation of pilot projects in organizations must account for human variation to evaluate new methods (and most does not)
- The most effective approaches to software development are capability based rather than process based
Summary

Defects

Human Variation

Uncertainty

Size

Official Top 7 Diagram
Four Core Influences

- Size
- Defects
- Uncertainty
- Human variation

Official Top 7 Diagram
Why Does This Matter?

- Each influence is important in itself
- Each influence interacts with each other influence, and the interactions are significant
- Goal: Help software professionals develop a mental model/framework for understanding current practices and new developments in software engineering
Latest Attempt to Explain Software Development

Software Engineering Essentials Lecture Series

CxLearn.com
Check it out now!
Construx Software is committed to helping individuals and organizations improve their software development practices. For information about our training and consulting services, contact stevemcc@construx.com
+1 (425) 636-0100

Construx

10900 NE 8th Street, Suite 1350
Bellevue, WA 98004
+1 (866) 296-6300
www.construx.com
class Class1 {

    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    /// <example>SThread
    static void Main(string[] args) {
        // Logon BTULicenseManager licenseManager = new BTULicenseManager();
        // put your valid license here
        string networkLicense = "00000000-0000-0000-0000-000000000000-00000000";
        string password = "your password here";
        licenseManager.SetLicense(networkLicense, password);
        Console.WriteLine("Your license was set successfully.");
        string snapPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + "\My Videos\South Park-(Freak Strike)-2004-09-17-0.mpg";
        BTULicenseManager.CreateLibrary(snapPath);
        Console.WriteLine("Your library was created.");

        // properties
        PUS.PropertyBag bag = new PUS.PropertyBag();
        bag.Properties = new ArrayList(new Dictionary<string, string>());
        bag.AddPropertyValue("EpisodeDescription", "The boys compete to appear on a talk show. (Edited by Beyond TV Framework)");
        bag.AddPropertyValue("Title", "South Park - Season 1"");
        bag.AddPropertyValue("Episode", "135");

        // Put the PUS.PropertyBag into a more friendly collection class.
        // It's a good idea for you to write a friendlier wrapper class that
        // would allow you to add and remove properties and cast back to
        // the PUS.PropertyBag type on the fly.
        ArrayList aProperties = new ArrayList(bag.Properties);
        aProperties.add(new Dictionary<string, string>(bag.Properties));

        // Change the "EpisodeDescription" property
        foreach (PUS.Property prop in aProperties)
        {
            if (prop.Name == "EpisodeDescription")
            { prop.Value = "The boys compete to appear on a talk show. (Edited by Beyond TV Framework)";
            }
        }

        // Create a new PUS.PropertyBag with the edited property
        PUS.PropertyBag newBag = new PUS.PropertyBag();
        newBag.Properties = aProperties.ToArray();

        // This method will edit the recording
        library.EditMedia(FullName, newBag);
        // Print properties to the console and verify the change
        Console.WriteLine("Edited properties of ", FullName);
        foreach (PUS.Property prop in bag.Properties)
        {
            Console.WriteLine("Property: ", prop.Name, prop.Value);
        }

        // Pause so you can see the output, hit enter to continue
        Console.WriteLine("Press any key to exit...");
        Console.ReadLine();
        return;
    }
}