



Simplifiers and Complicators

LCO Prototyping

CS 577a

Dan Port, USC

2001

Unmet Expectations Problems

- **LCO success condition**
 - Describes at least one feasible architecture
 - Satisfying requirements within cost/schedule/resource constraints
 - Viable cost-effective business case
 - Stakeholder concurrence on key system parameters
- **Projects That Failed LCO Criteria**
 - 1996: 4 out of 16 (25%)
 - 1997: 4 out of 15 (27%)

why?

Requirements and Expectations: Domain Model Clashes

- **Easy/hard things for software people**

“If you can do queries with all those ands, ors, synonyms, data ranges, etc., it should be easy to do natural language queries.”

“If you can scan the document and digitize the text, it should be easy to digitize the figures too.”

- **Easy/hard things for librarians**

“It was nice that you could add this access feature, but it overly (centralizes, decentralizes) control of our intellectual property rights.”

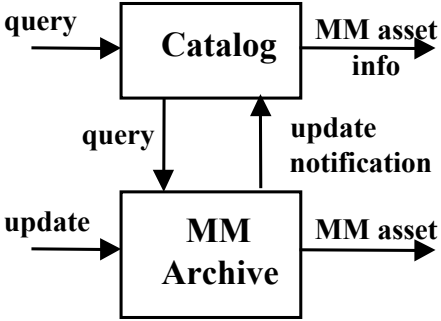
“It was nice that you could extend the system to serve the medical people, but they haven’t agreed to live with our usage guidelines.”



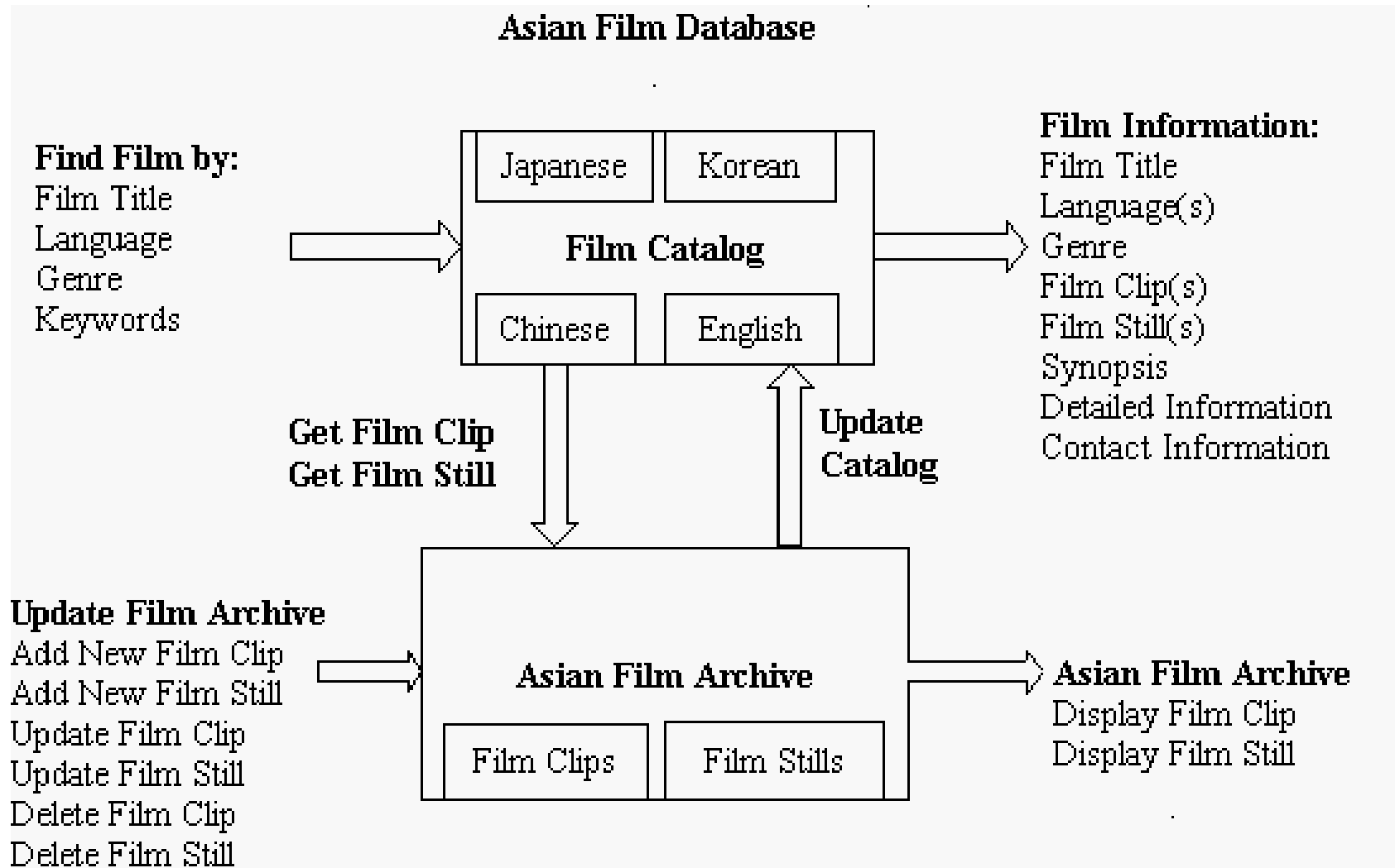
1998 Simplifier/Complicator Experiment

- **Identify application simplifiers and complicators**
 - For each digital library sub-domain
 - For both developers and clients
- **Provide with explanations to developers and clients**
 - Highlight relation to risk management
- **Homework exercise to analyze simplifiers and complicators**
 - For two of upcoming digital library projects
- **Evaluate effect on LCO review failure rate**

Example S&C's

Type of Application	Simple Block Diagram	Examples	Simplifiers	Complicators
<p align="center">Multimedia Archive</p>	 <pre> graph TD Q1[query] --> C[Catalog] C -- "MM asset info" --> O1[] C -- query --> A[MM Archive] A -- "update notification" --> C U1[update] --> A A -- "MM asset" --> O2[] </pre>	<p>1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 31, 32, 35, 36, 37, 39</p>	<ul style="list-style-type: none"> • Use standard query languages • Use standard or COTS search engine • Uniform media formats 	<ul style="list-style-type: none"> • Natural language processing • Automated cataloging or indexing • Digitizing large archives • Digitizing complex or fragile artifacts • Rapid access to large Archives • Access to heterogeneous media collections • Automated annotation/description/ or meanings to digital assets • Integration of legacy systems

Specialized S&C's



Simplifiers	Risks and Trade-offs
<p>Generic Uniform Media Formats</p> <p>Specific All video clips are stored using an open file format for video/audio (e.g., MPEG). All film stills are stored using an open image file format (e.g., JPEG). The inverse complicator is to store film clips using streaming video technologies</p>	<p>This means that we may have to convert existing digital assets or digitize the original media, which may be costly.</p> <p>A unique file format limits the user base to those who have viewers for that particular file format</p> <p>The chosen file format may not be the most efficient for the various types of media (in terms of compression rates, quality, etc...)</p>
<p>Generic Use Standard Query Languages</p> <p>Specific Organize catalog and archive relationally so that queries will be limited to standard search formats,; match exactly by value on any of the fields with or without using boolean combinations (AND, OR, NOT, etc...), or using pattern matching (SQL <i>LIKE</i> keyword)</p>	<p>May not be as effective for "discovering" assets in the archive: users must know what they're looking for, in order to search for it</p>
<p>Generic Use Standard COTS</p> <p>Specific Use a standard Relational Database Management System (RDBMS) that supports storing multi-media assets</p>	<p>A Relational Database Management System may not be most suited for archival of multi-media assets.</p> <p>A Relational Database Management System may have a high initial cost, high implementation, and high administration cost (requires specialized knowledge skills)</p>

Complicators	Risks and Trade-offs
<p>Generic Natural Language Processing</p> <p>Specific Store the information only in one language (e.g., English) and provide dynamic translation into Chinese, Japanese and Korean The inverse simplifier is to store the same information in 4 different languages (English, Chinese, Japanese and Korean).</p>	<p>The first approach is a complex, error-prone, expensive natural language processing issue</p> <p>The second approach will require more storage space, in addition to acquiring the translations</p>
<p>Generic Digitizing Large Archives</p> <p>Specific Digitizing film clips from the entire collection of films (which grows at a very fast rate of 800 films per year for Indian films alone)</p>	<p>If each film clip requires around 10 MB, then the rate of growth of the database will be of 8GB a year (exclusive of catalog information)</p>
<p>Generic Integration of "Legacy" Systems</p> <p>Specific Do not require Real-Video plug-in for Web browsers to allow users to view streamed film clips</p>	<p>We cannot use more effective multi-media formats, which are becoming standard technologies</p>

The Results

- **Projects That Failed LCO Criteria**
 - **1996: 4 out of 16 (25%)**
 - **1997: 4 out of 15 (27%)**
 - **1998: 1 out of 20 (5%)**
 - **1999: 1 out of 22 (4%)**
- **40% of Student critiques cited S&C's as helpful (and more since)**
 - **In focusing on achievable requirements set within tight schedule**
 - **In understanding project risks and tradeoffs**



Prototyping

Prototyping

- **Contents:**
 - **Goals of prototyping.**
 - **Types of prototype.**
 - **What should you have for LCO?**
 - **Prototyping guidelines.**
 - **To reuse to throw away?**
 - **Rapid prototyping tools.**
 - **How to choose a tool?**
 - **Scenarios.**

Prototyping goals, 1

- **Prototypes help with your customer negotiations:**
 - **Reality check: are you building what the customer expected?**
 - **A prototype gets you past “I’ll know it when I see it.”**
 - **Makes your project concrete for the customer.**
 - **Focuses negotiations on user concerns (when the customer isn’t the end user).**

Prototyping goals, 2

- **Prototypes help you design your product:**
 - Any gaps or inconsistencies in the design/requirements may be revealed.
 - Questionable or difficult aspects can be tried out.
 - Outright errors in your initial design may show up.
 - Weaknesses in the development team's skills may be revealed (in time to get training).
 - Unanticipated problems with implementation technologies may be revealed (in time to try something else).
 - More important or more difficult requirements or components show up; knowing about these things helps with making a reasonable schedule and division of labor.

Types of prototype, 1

- **Prototypes may be classified as:**
 - **Non-functional (for “look and feel”):**
 - Images.
 - Static interface (in some language).
 - Example interaction (series of slides, or a log or journal file).
 - **Functional (in various degrees):**
 - Anything that runs and shows off some system features.
- **Prototypes may be classified as corresponding to phases in the development, from “Initial” to “Pre-alpha”.***

* “Alpha” and “Beta” are industry parlance for pre-release software. An Alpha release includes major features, but isn’t intended for general use. A beta release should be mostly complete, but still needs testing.

Types of prototype, 2

- **Prototypes may be classified by their intended use:**
 - **A prototype might be used to demonstrate the user interface, rather than the program's function.**
 - **A prototype might be used to demonstrate a programs function (in this case the UI is less important).**
 - **Any test program written to “try out” a technology or design aspect is a prototype. Prototypes may exist only to help the development team, rather than to show to the world.**

Requirements, WinWin, Prototypes

- **Fundamental relationship between WinWin results, Prototypes, and Requirements**
- **No “formula” but some general rules of thumb**
 - WinWin agreements often become requirements
 - WinWin taxonomy items contribute to requirements
 - Specialized taxonomy (as in HW) can be used in SSRD sections
 - WinWin options may need to be prototyped
 - Prototype results may contribute to later WinWin cycles
 - Prototype results often contribute to requirements
 - Helps resolve IKIWISI requirements model clash
 - Iteration and simultaneous development common
 - Must integrate results and have some degree of consistency!

What should you have for LCO?

- **Your LCO prototype may be very simple!**
- **Your prototype should focus on demonstrating the core features/behaviors to your customer:**
 - **Your prototype doesn't have to be functional, necessarily, but should be visual enough to allow you to give your LCO audience (ie, your customer) a good idea of how the real thing will look and work.**
 - **A set of static web pages with forms is often a good first prototype.**
- **Focus on getting a “hardcore” prototype together for the LCA.**



LCO Prototyping guidelines (construction and presentation)

- **Create (and demonstrate) your prototype from the point of view of the user:**
 - Prototype should give an idea of what the real system will be like for the user
 - Use realistic data and use scenarios
 - Present highly technical or abstract design issues in a concrete manner
- **Focus on the main points first!**
 - What is the normal or most common user mode?
 - What will the most common user activity be?
 - What are the most important (core) features?
 - What are contentious issues/features among the development team, and with the customer?
 - What features will be difficult?
 - Leave optional, subtle or boring features for later.

To reuse or to throw away?

- **A key decision with a prototype is whether to try to make it reusable (such that it can be a basis for the construction of the final product), or whether to plan to throw it away. Consideration include:**
 - **Can you (or do you want to) do the prototype in the final implementation language?**
 - **How much time do you have for this prototype?**
 - **Are you planning multiple prototypes?**
 - **What is the goal for the prototype (UI, user behaviors, technological feasibility)?**

To reuse or to throw away?

- **Why would you throw away a prototype?**
 - In doing the prototype, the design or requirements change substantially.
 - Prototyping with an easy, fast tool lets you get on to the good parts.
 - Prototyping lets you try things easily, which may be difficult in the final implementation.
 - Prototype implementation likely did not follow development quality standards, be well commented, or have good documentation
- **Understand carefully what the tradeoffs are for evolving a prototype into the final product!**



Rapid prototyping tools

- **“Rapid prototyping tools” are languages or programs that let you create prototypes more quickly than you could in your project’s real implementation language/technology.**

Rapid prototyping tools

- **The languages/programs listed here may be prototyping tools for one project, but final implementation tools for another.**
- **Prototyping tools include (but are not limited to):**
 - Pencil and paper.
 - Images (Photoshop).
 - Static web pages (HTML and images).
 - Dynamic web pages (Javascript and CGI).
 - Scripting languages (Perl, Python, Tcl).
 - “Visual” programming environments (MS Visual Basic).
 - Database development tools (MS Access).
 - A “real” but “nice” language like Java (with AWT or Swing).

Tools: Static web pages

- **Web pages are provide a common, familiar look, and basic set of functions.**
- **May be used to demonstrate user input forms.**
- **May easily be made colorful and attractive.**
- **May be created in various visual editors (Netscape Composer, MS FrontPage, Adobe PageMill).**
- **Can be accessed almost anywhere, from any nearly any type of (networked) computer.**
- **May be easily augmented with a certain amount of functionality.**
- **Disadvantages:**
 - **Relatively static interface; the display is constrained to “pages”.**
 - **Input widgets are limited (relative to VB, for instance).**
 - **Layout varies across platforms.**

Tools: Dynamic web pages

- **Javascript can add considerable function to a web page:**
 - **Error and confirmation dialogues.**
 - **Simple processing (arithmetic).**
 - **Modifications to the display (eg, cycling an image).**
- **Frames can implement multi-window displays (eg, a “tool bar” and a “work area”).**
- **Server-side CGI can be used to implement complex processing (eg, database interaction).**

Tools: scripting languages

- **Benefits of a scripting language (Perl, Python, Tcl, sh, [Javascript]):**
 - Interpreted: you don't have to recompile.
 - High level data types (extensible arrays, hash tables).
 - High level system functions (file I/O, string manipulation, networking).
 - Regular expressions for parsing/scanning strings and files.
- **Tk is a windowing library that may be used with any of the above to quickly “script” a GUI.**
- **Disadvantages:**
 - Interpreted code is generally slower than compiled code.
 - Missing or flaky OO features (except Python).
 - Less support for proprietary libraries and existing code.

Tools: Visual programming environments

- **Visual Basic (and similar products such as Symantec Café) let you:**
 - Drag and drop to create an interface.
 - Define attributes and methods from menus.
 - Access lots of existing, reusable code.
- **VB is probably the very fastest way to put together a functional program with a sophisticated graphical user interface.**
- **Disadvantages:**
 - VB programs may be slower, larger and less robust than equivalent programs written with (say) Visual C++.
 - VB programs are in no way portable to a operation system other than windows.
 - VB programs may not be able access as many libraries and Windows internals as C++ programs can.

Tools: database development tools

- **Database development tools (such as MS Access) allow:**
 - Graphical design of a database and corresponding user interfaces (data entry, reports).
 - Access to different databases (you can “bring together” various existing databases).
- **These tools are particularly useful to try out the features of a complex database schema, particularly if you’re not very familiar with SQL.**
- **Disadvantages:**
 - MS Access isn’t portable off of Windows (and Mac?) platforms, though you may be able to use it to develop portable SQL statements.
 - Performance and external interfaces may be lacking (though not if you’re using, for example, SQL Server and IIS on Windows NT).

Tools: “real” languages

- You may find it convenient to code your prototype in a real language, like Java. Reasons include:
 - Your final product will be in Java.
 - You’re more familiar with Java (you can start using it now, while training on the implementation language).
 - AWT and Swing are flexible, powerful windowing libraries.
 - Java provides easier, safer networking and threads than C.
 - Java is type-safe and object-oriented.
- Disadvantages:
 - Programming in Java is usually slower than in, for example, Perl; there’s nothing “rapid” about prototyping in Java.
 - It takes more work to get something going in Java (particularly user interfaces); likewise it takes longer to change it, and thus it’s harder to try ideas.

How do I choose a prototyping tool?

- **Some considerations in choosing a prototyping tool:**
 - **Do you want to reuse the code?**
 - **Are you focusing on the user interface?
The behavior? Internal implementations?**
 - **How many prototypes are you doing?
Will you want to extend this one?**
 - **What tools do you know already?**
 - **What tools might be useful later (and are thus worth learning/practicing now)?**



Scenarios

- **The next few slides present three imaginary example project scenarios. Each describes the project, then describes a series of prototypes, their focus, and the tools they'll use.**

Scenario: A web application, 1

- **Project:** amazon.com, or a subset thereof. I.e., an e-commerce web site for selling books, with a shopping cart and credit card payment. The final system will be database-backed dynamic web pages via CGIs coded in Perl.
- **Prototype 1, technical feasibility:** a Perl program that makes credit card transactions work (probably by working with a library provided by an e-commerce service or bank. If this can't be made to work, the implementation language will have to be changed. If it still doesn't work, the project is doomed.

Scenario: A web application, 2

- **Prototype 2, basic UI:** a set of static web pages demonstrating the product search, product display, shopping cart contents, checkout and billing form screens look like. The page layouts will be there, but the team won't spend too much time on making them "pretty". The prototype will be modified iteratively in conjunction with the customer, until s/he is happy.
- **Prototype 3, full demo:** the pages from 2 are backed by functional scripts; a user may access the site, search for a product, add it to his/her cart, and then buy it. Credit transactions and inventory management may not be integrated yet. Again, the customer is consulted to make sure s/he approves of the interface and features.

Scenario: GUI application

- **Project:** A “slideshow” editor. The user can have a series of “slides” and switch back and forth between them. The user can add more or less arbitrary text to a slide. The system should be able to save slideshows to the disk and then open them again. The system should be in Java (for platform independence).
- **Prototype 1, UI:** A mock-up of the main application window in Visual Basic. It has all the buttons, tool bars, menus and main slide editing window, but they don’t do anything. The prototype is shown to the customer and a few of the future users, and refined according to their comments.
- **Prototype 2, editing a slide:** An rough start at the main slide-editing functionality, in Java. This prototype is for the use of the developers, to insure that they know how to do it, and to get an idea of the time required to finish the full project. Some code may be reuseable, but it will probably need to be redone.

Scenario: Application with no UI

- **Project: Implement a simple, fast web server in C. The server should support the full HTTP protocol, including support for CGI and cookies.**
- **Prototype 1, making sure we can code C: The first prototype implements a very basic server which takes client requests and answers 'OK'. The server has to do networking with sockets, has to be able to handle multiple requests simultaneously (threads or multi-process?) and has to be in C. This exercise proves the skills of the development team. The code may be reused if it is good; otherwise it will be redone in the construction phase.**
- **Prototype 2, getting the behavior right: The second prototype helps the team understand the the HTTP, CGI and Cookie protocols, so to implement them correctly. This prototype need not be fast, robust or handle multiple connections. The prototype will be in Perl or Python, for their easy, safe data structures and string processing and easy networking libraries. The final prototype should correctly implement the protocols, so to serve as an example to the real implementation in C.**

Exercise: design your own prototypes

- **Project: A web-based email client. ISD wants a full-featured email client on the web, so that students can check their USC email from anywhere. The system has to be extremely intuitive and easy-to-use.**
- **Step 1: Identify a set of requirements for the systems function and interface. Of these, which are most important (which have the highest priority)?**
- **Step 2: Suggest one or more prototypes to help in the following:**
 - **The customer negotiations (figuring out what ISD wants, what they mean by “full-featured”).**
 - **The design (how should the development team decide on an implementation? What are the hard parts of the implementation?).**
 - **The users perspective (how does the team develop a good, intuitive, easy-to-use interface?).**