

Requirements Engineering, Expectations Management, and the Two Cultures

Barry Boehm, Marwan Abi-Antoun, Dan Port, Julie Kwan, and Anne Lynch

University of Southern California

{boehm, marwan, dport}@sunset.usc.edu, {jkwan, annelync}@usc.edu

Abstract

One of the difficulties in requirements negotiation is to determine a feasible and mutually satisfactory set of requirements for the developer and the user, a problem related to C.P. Snow's "Two Cultures" problem. During the last year of our experience with an annual series of Digital Library projects, we have been experimenting with expectations management and domain specific lists of "simplifiers" and "complicators", as a way to address the "Two Cultures" problem involving librarians and computer scientists.

Initial results indicate that the simplifiers and complicators approach successfully reduced the number of projects having serious feasibility problems, and helped manage the expectations of both the developers and the customers/users. We see no obstacles to applying the approach to other domains.

1. Introduction

In his seminal work on the Two Cultures [1], C.P. Snow found that science and technology policymaking was extremely difficult because it required the combined expertise of both scientists and politicians, whose two cultures had little understanding of each other's principles and practices.

During the last three years, we have conducted over 50 real-client requirements negotiations for Digital Library applications projects. Those largely involve professional librarians as clients and 5-6 person teams of computer science graduate students as developers. We have found that the "Two Cultures" problem is one of the most difficult challenges to overcome in determining a feasible and mutually satisfactory set of requirements for these applications.

During the last year, we have been experimenting with expectations management and domain-specific lists of "simplifiers" and "complicators" as a way to address the two-cultures problem for software requirements within the overall digital library domain. Section 2 of this paper provides overall motivation and context for addressing the two-cultures problem and expectations management as significant opportunity areas in requirements engineering.

Section 3 discusses the digital library domain and our stakeholder Win-Win and Model-Based (System) Architecting and Software Engineering (MBASE) approach as applied to digital library projects. Section 4 discusses our need for better expectations management in determining the requirements for the digital library products over the first two years, and describes our approach in the third year to address the two-cultures problem via expectations management. Section 5 summarizes results to date and future prospects.

2. Motivation and Context

The two-cultures problem has been a particular source of difficulty within requirements engineering. Some interpretations of the waterfall model and the Total Quality Management "voice of the customer" guideline hold that the requirements should be determined by recording the customers' statements of need. However, if the customers have no idea of the relative cost and difficulty of a requirement, they are more likely to enter infeasible requirements as statements of need. Some examples of resulting requirements difficulties are:

- A commercial customer specified a natural language interface for an otherwise simple query system. The project was cancelled after the natural language interface caused a factor-of-5 overrun in project budget and schedule.
- A commercial customer specified a project to fully digitize a set of corporate records via scanning and optical character recognition. The resulting cost escalated by a factor of 10 after it was discovered that the records included many hard-to-capture tables, charts, and graphs.

Thus, one would like to find ways to achieve a reconciliation of customer expectations with developer capabilities before firmly committing to a set of requirements. A counterpart challenge is to ensure that the developer's interpretation of the requirements do not conflict with the users' operational concepts and human-computer interaction (HCI) styles. A frequent example involves programming experts' well-intentioned misapplication of the Golden Rule: *Do unto others* (create

a friendly HCI) *as you would have others do onto you* (create a programmer-friendly HCI).

The stakeholder win-win approach to requirements engineering [2] provides one way to achieve this reconciliation of expectations and capabilities. A hard-to-achieve customer or user Win Condition will conflict with the developer's Win Condition to minimize the risk of delivering an acceptable product within budget and schedule. In the win-win approach, this conflict is identified as an Issue needing resolution before the formal win-win equilibrium state (all Win Conditions covered by Agreements; no unresolved Issues) can be achieved [3].

The overall stakeholder WinWin negotiation approach is similar to other team approaches for software and system definition such as gIBIS [4], Viewpoints [5], GRAIL [6], Participatory Design and JAD [7]. Our primary distinguishing characteristic is the use of the stakeholder win-win relationship as the success criterion and organizing principle for the software and system definition process. Our negotiation guidelines are based on the Harvard Negotiation Project's techniques [8].

However, our experience with the Win-Win approach has taught us that this reconciliation is much easier if stakeholder expectations can be better managed prior to negotiations. One effective mechanism for doing this involves prototyping concurrently with negotiation [9]. For example, a government customer for a large transaction processing system initially insisted on a requirement for a 1-second response time. Meeting this requirement involved a custom architecture and a \$100M project. Subsequent prototyping showed that a 4-second response time was satisfactory for 90% of the transactions. With this performance requirement, a solution using commercially available technology was achieved for \$30M, with a 3-second response time.

A customer expecting a 4-second response time will consider a 3-second response time a satisfying win; a customer expecting a 1-second response time will consider a 3-second capability as a disappointing loss. See [10] for other perspectives on expectations management.

3. MBASE Approach to Requirements Engineering

Over our three years of developing digital library products for the USC Libraries, we have been evolving an approach called Model-Based (System) Architecting and Software Engineering (MBASE) [11]. MBASE involves early reconciliation of a project's success models (correctness, business case, stakeholder win-win, ...); product models (domain, requirements, architecture, ...); process models (waterfall,

evolutionary, spiral, ...); and property models (performance, reliability, ...). It extends the previous spiral model in two ways:

1. Initiating each spiral cycle with a stakeholder win-win stage to determine a mutually satisfactory (win-win) set of objectives, constraints, and alternatives for the system's next elaboration during the cycle.
2. Orienting the spiral cycles to synchronize with a set of life cycle anchor points: Life Cycle Objectives (LCO), Life Cycle Architecture (LCA), and Initial Operational Capability [12] (See Figure 1).

The LCO and LCA milestones involve the concurrent achievement of the following six system definition elements:

- An Operational Concept Description, including system goals and boundary, current system shortfalls and operational scenarios for the proposed system;
- Prototypes and/or key executing core capabilities of the system;
- A Requirements Definition, including capability, interface, quality attribute, and evolution requirements;
- An Architecture Definition, including some extensions of the Unified Modeling Language (UML) views and choices of COTS and reuse components;
- A Life Cycle Plan, including key software life cycle milestones, schedules, deliverables, organizational roles, and risk management;
- A Feasibility Rationale, including a business case analysis, and demonstrating satisfaction of the key success criterion: that a system built to the architecture will support the concept of operations, be compatible with the prototypes, satisfy the requirements, and be buildable within the budgets and schedules in the plan.

The primary success criteria for the LCO milestone are that the project has demonstrated a viable cost-effectiveness business case, has shown at least one architecture that can satisfy the Feasibility Rationale success criterion, and has achieved stakeholder concurrence on the key system parameters. The primary success criteria for the LCA milestone are that the project has committed to a specific architecture which satisfies the Feasibility Rationale success criterion, that the project has identified all its critical risks, and that these risks are either resolved or covered by a risk management plan.

In the MBASE approach, the Requirements Definition is developed concurrently with the other five system definition elements. Conflicts among the elements are

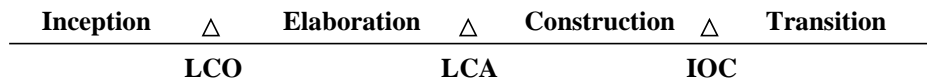


Figure 1: Rational/MBASE Phases and Milestones

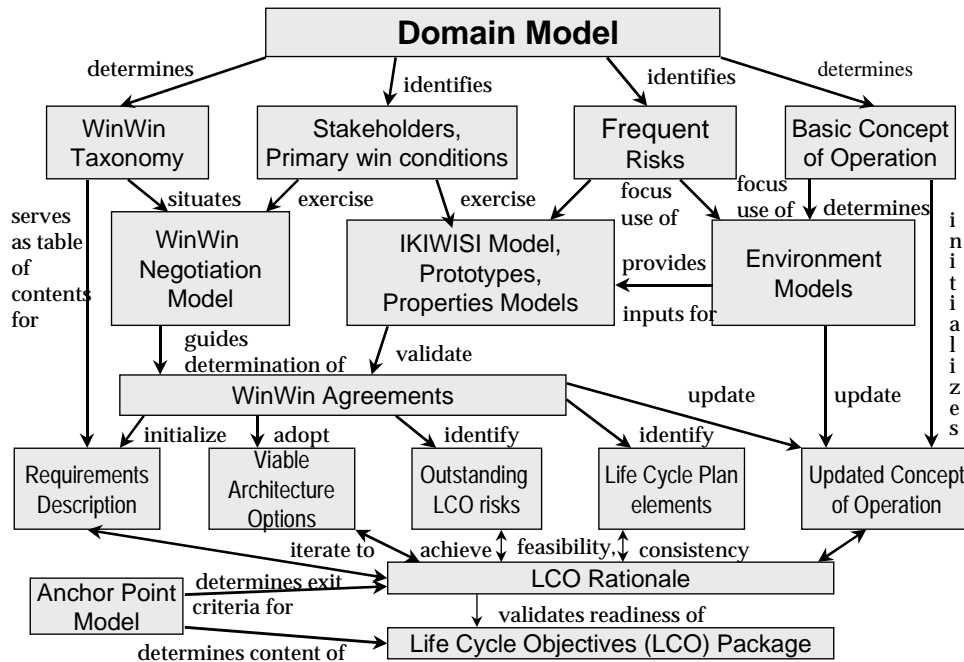


Figure 2: MBASE Model Integration: Inception Stage ending with the LCO milestone

resolved via the USC-WinWin tool, a multi-stakeholder groupware system, for requirements negotiation, including various tradeoff analysis tools. This concurrent engineering process uses one or more cycles of the WinWin Spiral Model [13] to achieve successful passage of the LCO and LCA anchor point milestones. MBASE acknowledges a strong dependence on the System Architecting approach developed in [14] and [15].

MBASE has co-evolved with the Rational Unified Process ([16], [17]), which has adopted the MBASE LCO, LCA, and IOC anchor point milestones as the boundaries between its Inception, Elaboration, Construction, and Transition phases; while MBASE has adopted the Rational phase-activity definitions (see Figure 1). In its approach to tradeoffs and satisficing among multiple quality objectives, MBASE is similar to the I* work of Yu ([18], [19]) and the SEI Architecture Tradeoff Analysis work [20]; see [21]. MBASE and User-Led Requirements Construction (ULRC) at the University of Manchester [22] are similar in considering user-led requirements engineering as an emergent, postmodern sociotechnical process.

For our digital library projects, we have roughly 11 weeks to go from a short problem statement (typically 4-5 sentences) to an LCA package, consisting of various prototypes and about 190 pages of documentation. This includes about 6 weeks for the Inception phase to develop the LCO version of the package. To reduce the risk that the various projects (typically 15-20) will go off in totally incompatible directions, we have developed a domain-

specialized Rapid Application Development of the MBASE approach for the digital library projects.

Figure 2 shows the MBASE approach for the Inception phase of the digital library projects. For the 1996 projects, the beginning point at the top of Figure 2 was a multimedia archive domain model furnished to the students. The domain model included the system boundary, its major interfaces, and the key stakeholders with their roles and responsibilities. The domain model also established a domain taxonomy, which was used as a checklist and

organizing structure for the WinWin requirements negotiation process.

As shown at the left of Figure 2, this taxonomy was also used as the table of contents for the requirements description, ensuring consistency and rapid transition from WinWin negotiation to requirements specification. The domain model also indicated the most frequent risks involved in multimedia archive applications. This was a specialization of the list of 10 most frequent software risks in [23], including performance risks for image and video distribution systems; risks that users could not fully describe their win conditions, but would need prototypes; and risks that users would expect more capabilities than could be developed in the 12-week spring-semester development period.

4. Experimental Approach to Expectations Management

The top-10 risk item checklist discussed above enabled the teams to identify some of the high expectations as risk items and to resolve them early. However, the risk items list is not sufficiently detailed or domain-specific to catch all the unrealistic expectations. As a result, during 1996 and 1997, we found that about 25% of the projects failed to satisfy the primary LCO Feasibility Rationale success criterion, i.e., to demonstrate at least one architecture that could satisfy the conditions in the Operations Concept Description, Requirements Description, and Life Cycle Plan. Following the LCO Architecture Review Board

feedback, these projects were able to produce satisfactory LCA packages, however with more effort and less success than if the problems had been detected earlier. The problems were due largely to the two-culture gap. For example, 4 out of the 15 projects in 1996 exhibited the following difficulties:

- A student film archive project focused on cataloguing and querying issues, and underestimated the performance problems involved in digital film distribution across a campus network.
- A multimedia museum project underestimated the range of variability required of the cataloguing, query, and distribution subsystems in dealing with heterogeneous media. The project assumed that a single set of generic subsystems would handle all of the media.
- A financial table reformatting system project underestimated the range of variability involved in dealing with a wide variety of heterogeneous word processing systems.
- A large photographic archive project focused on the issues of automating the archive's operation, and seriously underestimated the amount of effort required to digitize the photos.

In 1997, we had 4 of 16 projects exhibit similar LCO package difficulties:

- A Virtual Reference Librarian project underestimated the complexity of handling natural language queries, and of building artificial intelligence like capabilities to infer frequently asked questions from a set of natural language queries.
- A project to develop a front-end query capability for heterogeneous set of art-collection databases underestimated the complexity of developing a common query language and translator for the query inputs. The project did not address the even more difficult problem of handling the heterogeneous query outputs, including error responses.
- A famous-writer-archive project underestimated the difficulties of cleaning up archaic audiotapes, synchronizing audio and text, and providing an English-German multilingual capability.
- A project to convert legacy periodical records into a new set of common formats underestimated the complexity of creating a general set of record translators across a wide variety of periodical characteristics (variable publication frequency, volume numbering discontinuities, etc).

4.1. Simplifiers and Complicators: Overview

For the 1998 series of projects, we wished to reduce the frequency of these LCO package shortfalls. We

decided to test the following hypothesis: providing stakeholders with a better mutual understanding of application simplifiers and complicators will reduce overexpectations and the frequency of projects specifying unachievable requirements.

Our specific approach to test the hypothesis involved the following main steps to manage the digital library project stakeholders' expectations:

- For each of the major digital library application subdomains, we developed a characterization of the subdomain, including lists of simplifiers and complicators (S&C's) which would make the applications easier or harder to implement.
- We provided the S&C's to Library clients, with explanations in librarian terms by the Library project coordinators.
- We highlighted the S&C's in class lectures on risk management.
- We made the S&C's the subject of an early student homework exercise: to pick two candidate 1998 projects, to identify their subdomains, and then analyze their S&C's.
- We used "percentage of projects failing the LCO Feasibility Rationale success criterion" to measure the frequency of projects specifying unachievable requirements.

Given that projects, technology and student capabilities (e.g., with Web-based applications) change from year to year, and that other factors (e.g., personnel shortfalls or incompatibilities) may cause LCO package failures, this approach is necessarily imprecise. Also, it is incomplete as it does not test the relative effectiveness of S&C's vs. prototypes or other methods of expectations management. However, the number (15-20 per year) and similarity of the projects provides a stronger-than-usual opportunity to test hypotheses about real-client requirements engineering.

4.2. Simplifiers and Complicator (S&C) Tables

The S&C tables by digital library subdomain are presented in Table 1. An explanation of the initial row (Multimedia Archive) is given below. The first column in Table 1 provides a typical top-level block diagram for a multimedia archive. Users can query the Catalog for multimedia asset information. When they find assets of interest, they can query the multimedia Archive to receive electronic copies of the assets. Administrators can update the Archive, and transmit update notifications for the Catalog. The second column identifies the multimedia archive projects which have been done to date. The list of project titles by number is provided at the end of Table 1. Many of these projects have artifacts accessible via the class web site (<http://sunset.usc.edu/classes/>).

Table 1. Developer-Side Simplifiers and Complicators

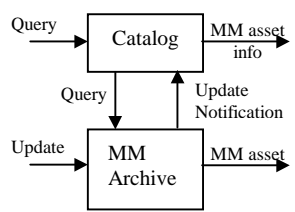
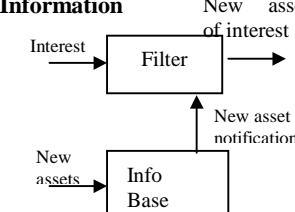
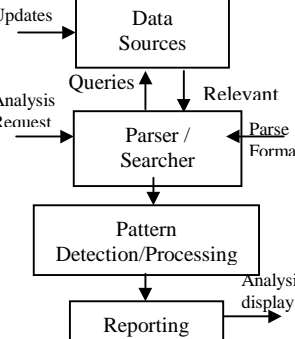
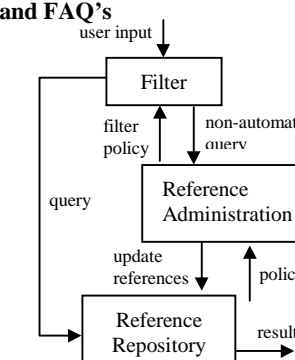
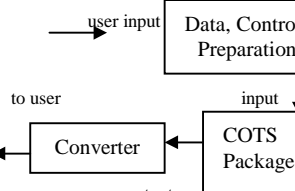
Sub-Domain/ Block Diagram	Example	Simplifiers	Complicators
<p>Multimedia Archive</p>  <pre> graph TD Catalog[Catalog] -- "MM asset info" --> MMArchive[MM Archive] MMArchive -- "Update Notification" --> Catalog Catalog -- "Query" --> MMArchive MMArchive -- "Update" --> Catalog MMArchive -- "Update" --> Out[] Catalog -- "Query" --> In[] </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
<p>Selective Dissemination of Information</p>  <pre> graph TD InfoBase[Info Base] -- "New asset notification" --> Filter[Filter] Filter -- "New assets of interest" --> Out[] Filter -- "Interest" --> In[] </pre>	<p>38, 41</p>	<ul style="list-style-type: none"> • Use of existing or standard information base • Well defined distribution points • COTS notification and event processing • WWW/internet based • Restricted interests vocabulary and filtering structures • Single information base 	<ul style="list-style-type: none"> • Volatile or ill-defined interest or filtering criteria • Complex distribution • Multiple distribution formats • Heterogeneous information sources • Complex filter reasoning • Automated interest update
<p>Data Analysis and Acquisition</p>  <pre> graph TD DS[Data Sources] -- "Updates" --> DS DS -- "Queries" --> PS[Parser / Searcher] PS -- "Relevant" --> DS PS -- "Parse Format" --> PS PS --> PDP[Pattern Detection/Processing] PDP --> R[Reporting] R -- "Analysis display" --> Out[] </pre>	<p>23, 28, 29, 43, 44</p>	<ul style="list-style-type: none"> • Use of data analysis packages (statistics, etc.) • Implement using interpreted or script languages (e.g. PERL) • Data stored in an RDBMS • COTS reporting packages (for graphics, etc.) • Simple, consistent data formats 	<ul style="list-style-type: none"> • Natural language processing • Highly unstructured data • High degree of formatting or conversion • Computationally intensive reporting • Spatial data analysis • Complex pattern recognition
<p>Automated Reference Services and FAQ's</p>  <pre> graph TD Filter[Filter] -- "filter policy" --> RA[Reference Administration] RA -- "non-automated query" --> RR[Reference Repository] RR -- "results" --> RA RA -- "update references" --> Filter Filter -- "query" --> RR </pre>	<p>18, 24, 28, 33, 34, 45</p>	<ul style="list-style-type: none"> • Single repository • COTS repository • WWW interface • Restricted query vocabulary and format • Stable FAQ's 	<ul style="list-style-type: none"> • Natural language processing of query • Multiple domains • Heterogeneous repositories • Complex or volatile filter policies • Volatile FAQ's
<p>COTS Package Extension</p>  <pre> graph TD DCP[Data, Controls Preparation] -- "input" --> CP[COTS Package] CP -- "output" --> Conv[Converter] Conv -- "to user" --> Out[] </pre>	<p>2, 23, 27, 30</p>	<ul style="list-style-type: none"> • Clean, well-defined API's • Single COTS package • Simple mappings of interface inputs and outputs 	<ul style="list-style-type: none"> • Dynamic API's • Natural language processing • Multiple, incompatible COTS packages • Complex exception handling • Volatile COTS packages

Table 1. Developer-Side Simplifiers and Complicators (continued)

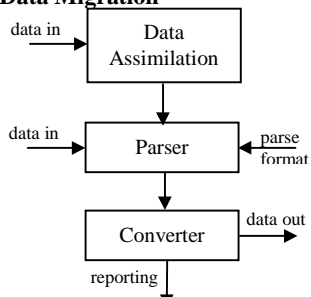
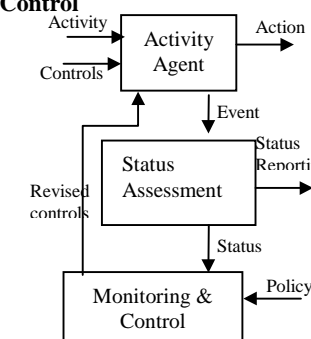
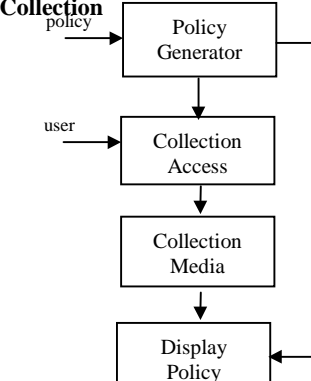
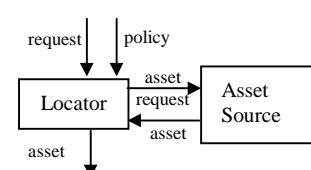
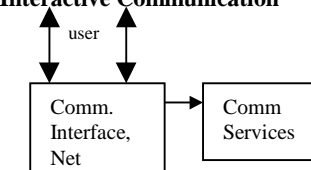
Sub-Domain/ Block Diagram	Example	Simplifiers	Complicators
<p>Data Migration</p>  <pre> graph TD DI1[Data Assimilation] --> P[Parser] DI2[Parser] --> C[Converter] C --> RO[reporting] DI3[Data in] --> DI1 DI4[Data in] --> P PF[parse format] --> P C --> DO[data out] </pre>	<p>2, 22, 29</p>	<ul style="list-style-type: none"> • Single data source; uniform data • Simple, stable parse formats 	<ul style="list-style-type: none"> • Heterogeneous or distributed data sources • Context sensitive conversions • Large number of exceptions • Highly relational data • Complex computational dependencies
<p>Activity Monitoring and Control</p>  <pre> graph TD AA[Activity Agent] --> A[Action] AA --> E[Event] SA[Status Assessment] --> AA SA --> SR[Status Reporting] MNC[Monitoring & Control] --> SA MNC --> P[Policy] SA --> RC[Revised controls] RC --> AA AA --> AC[Controls] AA --> Act[Activity] </pre>	<p>46</p>	<ul style="list-style-type: none"> • Standards based agent interfaces • Simple, well-defined policies • Uncoupled controls 	<ul style="list-style-type: none"> • Real time or embedded monitoring • Synchronization of monitoring & control activities • Concurrency management of activities • Distributed monitoring of activities • Natural language processing of policy • Policy learning
<p>Virtual Access to Special Collection</p>  <pre> graph TD PG[Policy Generator] --> CA[Collection Access] CA --> CM[Collection Media] CM --> DP[Display Policy] DP --> PG U[user] --> CA P[policy] --> PG </pre>	<p>19, 40</p>	<ul style="list-style-type: none"> • Fixed virtual architecture • Stable collection 	<ul style="list-style-type: none"> • Non-homogenous physical architecture • Immersive virtual reality (VR) • Heterogeneous and/or volatile collections
<p>Distributed Borrowing</p>  <pre> graph LR L[Locator] --> AS[Asset Source] AS --> L R[request] --> L P[policy] --> L L --> A[asset] AS --> AR[asset request] AR --> AS </pre>	<p>16, 42</p>	<ul style="list-style-type: none"> • Homogeneous asset sources • Simple asset source interfaces • Few asset sources 	<ul style="list-style-type: none"> • Complex borrowing policies and requests • Organization politics and economics
<p>Interactive Communication</p>  <pre> graph TD U[user] <--> C[Comm. Interface, Net] C --> CS[Comm Services] </pre>	<p>21, 25</p>	<ul style="list-style-type: none"> • Internet/WWW interface • COTS communication services 	<ul style="list-style-type: none"> • High degree of integration • Real-time • Synchronous • Concurrency • Rich media (video, voice, NLP, etc.)

Table 1. Developer-Side Simplifiers and Complicators (continued)

List of Examples

1996-97

1. Cinema-TV Moving Images
2. EDGAR Corporate Data
3. Hancock Library Image Archive
4. ITV Course Material
5. Korean-American Museum
6. Latin American Pamphlets
7. Digital Maps
8. Medieval Manuscripts
9. Planning Documents
10. Photographic Archives
11. Stereoscopic Slides
12. Technical Reports Archive

1997-98

13. Architecture & Fine Arts Databases
14. Bella Lewitsky Archives

15. Business School Working Papers
16. Inter-Library Loan
17. Engineering Technical Reports
18. General Library FAQ's
19. Hancock Museum Virtual Tour
20. Lion Feuchtwanger Archive
21. Network Consultation Support
22. Serial Control Records
23. Statistical Charts
24. Education Reference Assistant
25. Cross-Cultural Teaching Model
26. Business Reference Assistant
27. Online Catalog Search Strategies
28. Forms development for data ingest
29. Forms Development

1998-99

30. Data Mining the Library Catalog

31. California Virtual University Database: USC Entry Database
32. Dissertations
33. Education Reference Assistant
34. Business Q&A's
35. Asian Film Database
36. WWI - Record Enhancement
37. Digital Document Creation
38. Current Awareness Service for Social Work Doctoral Students
39. Hispanic Digital Archive
40. Book Locator for Doheny Stacks
41. New Booklist
42. Web Document Delivery
43. Data Input Into the Digital Library
44. Voice Metadata Ingest
45. Authoring tool for ADE system
46. Seaver Auditorium Scheduling

Table 2: Client-side Simplifiers and Complicators

Simplifiers	Complicators
<ul style="list-style-type: none"> • Project scope fits within client's authority scope 	<ul style="list-style-type: none"> • Scope crosses organizational boundaries
<ul style="list-style-type: none"> • Solution reduces job tedium, reduces procedural delays 	<ul style="list-style-type: none"> • Solution creates more user work, dehumanizes personal interactions
<ul style="list-style-type: none"> • Solution reduces organizational friction, infrastructure clashes 	<ul style="list-style-type: none"> • Solution shifts power, confuses lines of authority, puts outside parties on critical path
<ul style="list-style-type: none"> • Task-tailorable user interface 	<ul style="list-style-type: none"> • Mismatches between user interface and user tasks, capabilities
<ul style="list-style-type: none"> • COTS product features anticipate direction of growth 	<ul style="list-style-type: none"> • COTS product features evolving toward different marketplace
	<ul style="list-style-type: none"> • Hidden costs: licenses, data entry, conversion
	<ul style="list-style-type: none"> • Mismatches with existing legacy-system constraints
	<ul style="list-style-type: none"> • Single-criterion optimization: speed; correctness
	<ul style="list-style-type: none"> • Creeping (baroque) elegance

The third column identifies project decisions which would generally simplify the development of the archive capability. Their use needs to be balanced with other project considerations. For example, some standard query languages are not very user-friendly, but it may be better to provide a user-friendly front end to the standard package than to develop a totally new query capability. The list of simplifiers will necessarily be dynamic. For example, some projects this year are experimenting with the IBM Digital Library COTS package. If it meets USC Library acceptance criteria, its use will become a major simplifier.

The final column in Table 1 identifies project decisions which would generally complicate the development of the archive. Examples from the lists of 1996 and 1997 LCO problem projects included digitizing large archives, rapid access to large archives,

access to heterogeneous media collections, and automated annotation (synchronization) of digital assets.

In addition, we developed an initial list of client-side simplifiers and complicators (Table 2). These are conditions that make the library clients' job easier or harder, which are generally beyond the awareness of computer science specialists. For example, we have had some student teams add what they thought would be helpful extensions to their systems -- and to be totally surprised when their clients told them that the extensions made the system less useful, because they created dependencies on organizations outside the client's scope of control.

We should also mention that the two cultures situation also frequently produces pleasant surprises. These happen when the library client does not ask for capabilities because they appear too difficult, but the

student teams bring these up as candidate improvements. For example, the second 1996 team presented with the financial table reformatting problem identified a number of Web-based extensions to simplify querying for the sources of the tables, and to provide links to related Web-available information such as the Web home page of the company being analyzed. The client was amazed at how much more powerful a capability she was able to obtain in the project's short duration [13].

4.3. Project Results

Only one of the 20 1998 projects failed its LCO Feasibility Rationale success criterion. This is a 5% failure rate compared to the 25-27% failure rates in 1996 and 1997.

Below is an example S&C analysis for one of the 1998 projects in the Multimedia Archive sub-domain: the Asian Film Database. Figure 3 shows one instantiation of the general Multimedia Archive sub-domain block diagram. The archive's purpose is to provide Internet-based access to a collection of roughly 1200 films per year produced in China, India,

Japan, Korea, and Taiwan. Desired capabilities included multilingual query and viewing of film clips and still images. Tables 3 and 4 show the S&C analyses, as they applied to the Asian Film Database Archive project.

Having the students perform such analyses as pre-project homework and having the librarians exposed to the nature of the S&C's increased the teams' ability to identify and avoid overexpectations.

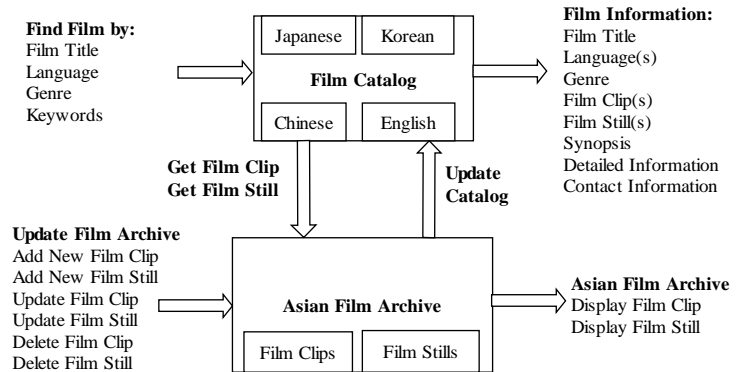


Figure 3: Asian Film Database Block Diagram

Table 3: Asian Film Database Simplifiers Analysis

Specific Simplifier	Risks and Trade-offs
Uniform Media Formats 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	This means that we may have to convert existing digital assets or digitize the original media, which may be costly. A unique file format limits the user base to those who have viewers for that particular file format. The chosen file format may not be the most efficient for the various types of media (in terms of compression rates, quality, etc...)
Use Standard Query Languages Organize catalog and archive relationally so that queries will be limited to standard search formats	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
Use Standard COTS Use a standard Relational Database Management System (RDBMS) that supports storing multi-media assets	A Relational Database Management System may not be most suited for archival of multimedia assets, may have high initial, implementation, or administration costs

Table 4: Asian Film Database Complicators Analysis

Specific Complicator	Risks and Trade-offs
Natural Language Processing 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).	The first approach is a complex, error-prone, expensive natural language processing issue. The second approach will require more storage space, in addition to acquiring the translations.
Digitizing Large Archives 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)	If each film's clips require around 100 MB, then the rate of growth of the database will be of 80 GB a year (excluding the size of the metadata or catalog information)
Integration of "Legacy" Systems Do not require Real-Video plug-in for Web browsers to allow users to view streamed film clips, as legacy systems do not support them.	We cannot use more effective multi-media formats, which are becoming standard technologies

5. Conclusions

Expectations management is a valuable technique in requirements engineering. It helps stakeholders converge more rapidly and effectively on mutually satisfactory and achievable system requirements.

The Two Cultures problem creates major expectations management challenges, since software developers and users generally only weakly understand what is easy or hard for each other to do.

The Simplifiers and Complicators approach reduced expectations management and requirements problems. For the 1998 series of digital library/applications, the approach successfully reduced the number of projects having serious Life Cycle Objective milestone feasibility problems, from 4 in 15 for 1996, and 4 in 16 for 1997, to 1 in 20 for 1998.

We see no significant obstacles to applying the approach to other domains and development situations. We plan to refine the approach for our 1999 digital library projects, and to experimentally apply it in other domains.

Software engineering education should include experience with non-software clients. Project courses developing compilers, operating systems, tools, or infrastructure middleware leave students unprepared for the Two Cultures problem many of them will encounter in their future careers.

12. References

- [1] C.P. Snow, *The Two Cultures and the Scientific Revolution*, Cambridge University Press, 1959.
- [2] B.W. Boehm, P Bose, E. Horowitz, M.J. Lee, "Software Requirements As Negotiated Win Conditions", *Proceedings of ICRE*, April 1994, pp.74-83.
- [3] M.J. Lee, Formal Modeling of the WinWin Requirements Negotiation System, *Ph.D. Thesis*, USC Computer Sciences Dept., 1996.
- [4] J. Conklin and M. Begeman, "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," *ACM Trans. OIS*, October 1988, pp.303-331.
- [5] A. Finkelstein, J. Kramer, B. Nusibeh, L. Finkelstein, and M. Goedicke, "Viewpoints: A Framework for Integrating Multiple Perspectives in System Development," *International J. Software Engineering and Knowledge Engineering*, March 1992, pp. 31-58.
- [6] A. Dardenne, S. Fickas, and A. van Lamsweerde, "Goal-Directed Concept Acquisition in Requirement Elicitation," *Proceedings, IWSSD 6, IEEE*, October 1991, pp. 14-21.
- [7] E. Carmel, R. Whitaker, and J. George, "PD and Joint Application Design: A Transatlantic Comparison," *Comm. ACM*, June 1993, pp. 40-48.
- [8] R. Fisher and W. Ury, *Getting To Yes*, Houghton-Mifflin, 1981.
- [9] B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, "Using the WinWin Spiral Model: A Case Study," *IEEE Computer*, July 1998, pp. 33-44.
- [10] N. Karten, *Managing Expectations*, Dorset House, New York, NY, 1994.
- [11] B. Boehm and D. Port, "Escaping the Tar Pit: Model Clashes and How to Avoid Them," *ACM Software Engineering Notes*, January 1999, pp. 36-48.
- [12] B. Boehm, "Anchoring the Software Process," *IEEE Software*, July 1996, pp. 73-82.
- [13] B. Boehm, A. Egyed, J. Kwan, and R. Madachy, "Developing Multimedia Applications with the WinWin Spiral Model," *Proceedings, ESEC/FSE 97*, Springer Verlag, 1997.
- [14] E. Reichtin, *Systems Architecting*, Prentice Hall, 1991.
- [15] E. Reichtin and M. Maier, *The Art of System Architecting*, CRC Press, 1997.
- [16] W.E. Royce, *Software Project Management: A Unified Framework*, Addison Wesley, 1998.
- [17] P. Kruchten, *The Rational Unified Process*, Addison Wesley, 1999.
- [18] E. Yu, "Modeling Strategic Relationships for Process Reengineering", *Ph.D. Thesis*, U. of Toronto, 1995.
- [19] E. Yu, and J. Mylopoulos, "Understanding 'Why' in Process Modeling, Analysis, and Design", *Proceedings, ICSE 16, IEEE/ACM*, May 1994, pp. 159-168.
- [20] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison Wesley, 1998.
- [21] B. Boehm and H. In, "Identifying Quality Requirements Conflicts", *IEEE Software*, March 1996, pp. 25-36.
- [22] D. Flynn, *Information Systems Requirements: Determination and Analysis*, 2nd Edition, McGraw-Hill, 1997.
- [23] B. Boehm, *Software Risk Management*, IEEE-CS Press, 1989.