SYST 542
Decision Support Systems Engineering

Instructor: Kathryn Blackmond Laskey

Fall Semester, 2006

Unit 7: Putting the Pieces Together: The DSS Lifecycle

Outline

- DSS Lifecycle
  - Iterative lifecycle models
  - Activities at each stage of lifecycle
- Requirements definition
- Integration of components
- Tools and generators
- Deploying DSS

Basic Realities of DSS Design

- **Requirements cannot be pre-specified**
- **Communications gaps are inevitable, and should be planned for**
  - Users are problem experts; engineers are technology experts
  - Users have implicit knowledge; engineers require explicit specifications.
- **Motivation barriers often exist**
  - It's hard to appreciate benefits of a system you can't see
  - Users may be gun-shy from overselling
  - Engineers may not appreciate the user's problem
- **Extensive iteration is both necessary and desirable**

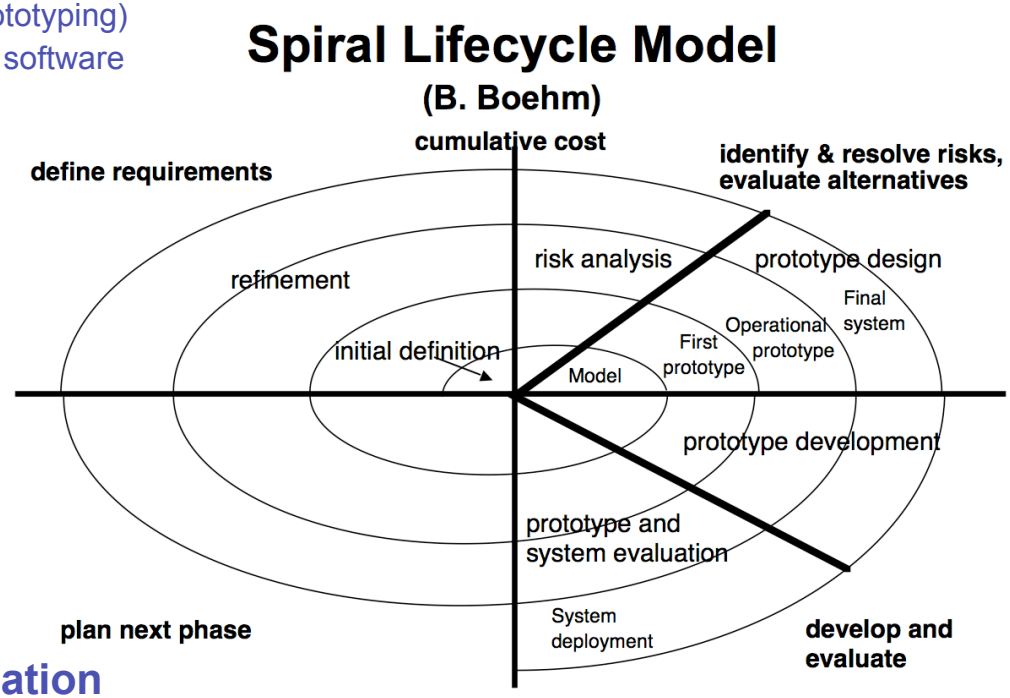
DSS Design - An Evolutionary Process

- **Iteration and feedback built into design cycle**
  - Requirements evolve through design and development
  - Prototyping is essential communication tool between developers and users
- **Often evolution continues even after deployment**
  - Additional functionality
  - Changes in user needs
  - Evolution in HW and SW capability

DSS Development Process

1. **Requirements analysis**
2. **Preliminary conceptual design**
   - system modeling (storyboarding, early prototyping)
   - option generation for methods, hardware, software
3. **Logical design and architectural specifications**
   - methods selection
   - software selection
   - hardware selection
4. **Detailed design and testing**
   - detailed software design
   - hardware configuration
5. **Operational implementation**
   - implement custom components
   - integrate COTS components with custom components
6. **Evaluation, feedback, and modification**
7. **Operational deployment**
   - system packaging and transfer

(Sage, 1991 & Andriole, 1990)

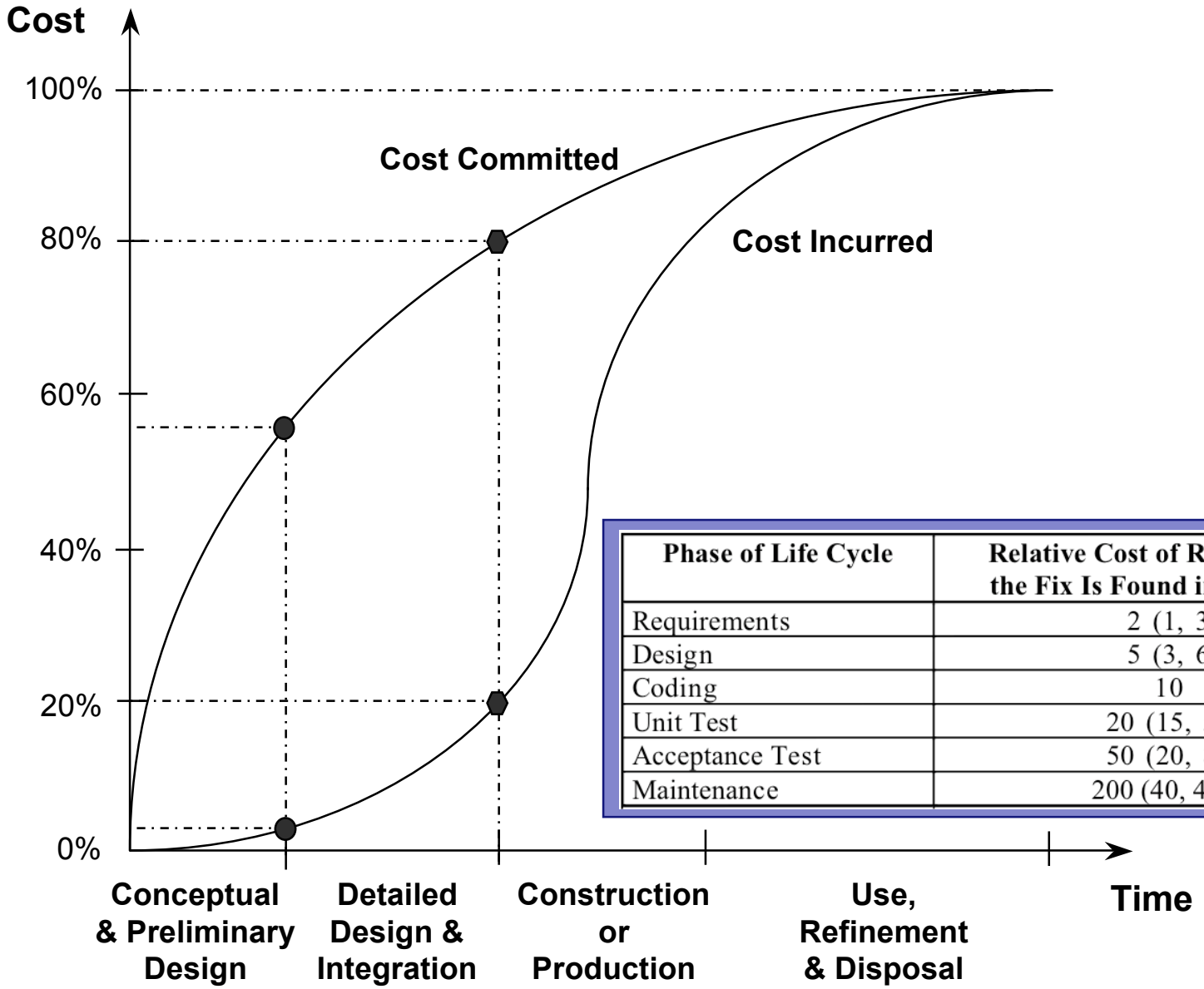


System Lifecycle Engineering

- **Stages of lifecycle**
  - ***Plan***: requirements identification, functional decomposition, preliminary design
  - ***Develop***: detailed design and implementation
  - ***Operate and evaluate***: test and evaluation, installation, operation and maintenance
  - ***Disposal***
- **Engineering process is applied within each stage**
  - ***Decomposition***: break into manageable sub-problems
  - ***Analysis***: solve the subproblems
  - ***Synthesis***: put the solutions together into a solution to the global problem
- **Systems engineering involves both technical and management aspects**
  - ***Technical***: what needs to be done
  - ***Management***: who does it, when to do it, resource (cost, schedule and other) constraints

Requirements Analysis: The Weak Link

60 to 80% of all system errors can be traced to inaccurate requirements definitions. (Boar, 1984)



Phase of Life Cycle	Relative Cost of Repair Given the Fix Is Found in this Phase
Requirements	2 (1, 3)
Design	5 (3, 6)
Coding	10
Unit Test	20 (15, 50)
Acceptance Test	50 (20, 80)
Maintenance	200 (40, 400)

Goal of Requirements Identification

- Determine decisions to be supported
- Identify functions to be performed
- Allocate functions to user/DSS
- Develop preliminary feasibility assessment

Role of User

- Users should be <u>active members of the design team</u>
- Advantages of active user participation
  - Designer exposure to user needs
  - User exposure to design process
  - System advocate within user community
  - Facilitate communication between users and system designers
- What happens if you don't involve users
  - Solving the wrong problem
  - Building a system that is not usable
  - Leaving out important functionality
  - "Orphan system" - no ownership

Staged Prototyping

- Develop initial model of system
- Select subproblem for early implementation
- Implement and evaluate DSS for subproblem
- Revise model of system
- Iteratively increase functionality of system

Benefits of Prototyping

- **System definition evolves incrementally**
  - Plan explicitly for iteration
  - Capture initial set of needs and implement quickly
  - Refine as mutual user / developer understanding evolves
- **Software prototype focuses user/developer communication**
- **Facilitates identification and control of project risk**
  - Reduces risk of wasting resources solving the wrong problem
  - Enables identification of problems early, when modification is less expensive

How Prototyping Can be Misused

- **Prototype the easy parts to show quick progress**
  - Prototyping should be a risk mitigation strategy
  - Prototype should focus on high-risk items to identify and mitigate risks
- **Storyboard functions that can't be implemented**
  - It's easy to draw pretty pictures but putting something behind them may be very hard
- **Be careless: It's a throwaway prototype and it doesn't really matter if it's right**
  - Good prototyping is just as disciplined (but in a different way) as traditional design
- **Overpolish the prototype**
- **Let users think the prototype is the system**
  - It often happens
  - It's designer's responsibility to make sure users understand purpose & role of prototype

Empirical Evaluation of Prototyping

- **Conventional development methodologies and prototyping compared on same problem (e.g., Boehm, et al)**
- **Advantages of prototyping**
  - systems easier to learn and use
  - development less affected by deadline pressures
  - prototypers' code 40% as large
  - prototypers accomplished task with 45% less effort
  - enhanced communication with users
  - increased user satisfaction and acceptance of systems
- **Pitfalls of prototyping**
  - don't use prototyping without enlisting cooperation from all affected parties
  - planning and scheduling may be difficult to fit into organizational culture
  - reduction in programmer and management discipline
  - thinking the prototype is the system
    - » overselling and overpromising
    - » resistance to continued development
  - temptation to stub difficult parts of functional design without first understanding design requirements

Inputs to Requirements Identification

1. Task profile
2. User profile
3. Organizational/doctrinal profile
4. Task/user/organizational interactions

Task Profiling

- **Goal: Understand process well enough to:**
  - Prioritize tasks to be supported
  - Design computerized support
- **Tools: Job and Task Analysis**
  - Job Analysis - The process of making a detailed study of a job to determine the duties, facilities required, conditions of work, and qualifications needed for its performance
  - Task Analysis - The process of identifying all the things that must be done to complete an activity satisfactorily
- **Sources of information:**
  - Reading materials
  - Surveys and questionnaires
  - Interviews
  - Observation
  - Simulation and gaming methods

Should You do a Formal Job&Task Analysis?

- Best route to complete, thorough, correct characterization of process to be supported
- Doing it "right" is:
  - Highly structured
  - Time consuming
  - Tedious
- What do need for your application?
  - Full-scale analysis may be neither necessary nor feasible
  - But you get what you pay for
    - » *Careless job&task analysis can doom your system*!
    - » If you don't have the time do do it right now, you'll have to make the time to do it over later!

User Profiling

Goal: characterize users on dimensions relevant to DSS requirements

Relevant dimensions:

- **Experience**
  - » task
  - » computers
  - » analytical methodology
- **Attitudes and expectations for DSS**
- **Cognitive abilities and style**

  visual/verbal

  analytical/intuitive

Sources of information: same as task profiling

Organizational/Doctrinal Profiling

Goal: Characterize organizational mission and doctrine to determine how proposed DSS fits into overall organizational context

Approach:

- identify organization's critical activities
- test compatibility of candidate DSS approaches

Methods:

- interviews
- policies & procedures manuals
- observations

Important Issues in Organizational/Doctrinal Profiling

- What are the critical activities of the organization?
- What is the hierarchical structure of the organization?
- What are the standard modes of interaction of individuals in the organization?
- How are problems solved within the organization?
- How does information flow within the organization?
- What is the regulatory climate in which the DSS will be used?
- How will the DSS fit in?

Preliminary Feasibility Analysis

- **Identify options**
  - off-the-shelf commercial DSS
  - "quick-fix" DSS from a DSS generator
  - large-scale DSS development
- **Estimate costs of each option**
- **Evaluate benefits of each option**
- **Evaluate risks of each option**
  - develop "reasonable worst case" scenarios
- **Evaluate options and select option**

DSS Modeling

Goal: Develop a model which captures the essence of the system-to-be

- facilitate discussion about function and role of system
- identifies weak points, inconsistencies, high-risk areas

Methods:

- narrative
- flowcharting methods
- storyboards

DSS modeling is different from decision problem modeling!

Storyboards

- Walk user through hypothetical screen displays
- On-line storyboarding tools are becoming increasingly common
- Facilitates user reaction by giving users "look and feel"
- Rapid modification of on-line storyboard facilitates iterative requirements redefinition
- Can be used in conjunction with rapid applications prototyping
- Evolving storyboard with annotations provides a record of the system development process

Functional Modeling

- Define functions system must perform
- Define inputs, outputs and flows
- Begin with high-level functions and decompose into lower levels until functions can be allocated onto software modules and/or user activities
- Diagrammatic methods such as IDEF0 are useful

Methods Selection: Factors to Consider

- Match to task
- Difficulty of implementing
  - available in generator?
  - available in toolkit?
- Difficulty of integrating
  - hybrid models may require integrating applications from different vendors
  - may have different internal representations and file formats
- Data requirements
- Computational requirements
- Understandability/user acceptance
- DSS builder competence

Challenge: Data Synchronization

- Data subsystem may draw data from multiple primary sources
  - Relational databases, OO databases, web sites
- Sources may have different semantics, different data models
- Extracted data must be kept current with source repositories

The Model-Data Link

- **Models draw input data and parameter values from the database** - DBMS is responsible for validity and edit checking
- **All models draw on the same database** - ensures consistency and currency
- **Models put output data and values back into database** - models use outputs from other models as inputs; data-dialog link used to display results
- **Models can be stored as data**

Issues with Model-Data Link

- Understanding model API and/or file format
- Translating data into a representation that can be understood by model
- Dealing with required but unavailable model inputs
  - DSS development may require a model to estimate inputs to other models from available information
- Translating model output into a representation that can be stored in DB and/or used by other models

The Model-Dialog Link

- **Invocation (D-->M)**
- **Parameter request (M-->D)**
- **Parameter collection (D-->M)**
- **Interrupt (D-->M)**
- **Notification (of completion or interrupt) (M-->D)**

  (note: results are placed in DB and passed via data-dialog link)

Issues with Model-Dialog Link

- Obtaining required inputs from user without disrupting workflow
- Control
  - Human-initiative
  - Computer-initiative
  - Mixed-initiative
- Responding to changes in environment and workflow
  - Changing initiative depending on environmental conditions
- Representing and presenting outputs in a way that makes decision relevance clear

The Data-Dialog Link

- Present model results to user
- Transform data and model results into user representations
- Process user requests for data retrieval and display (access)
- Request needed data from user (update)

Integrating the Components: Potential Difficulties

(Sprague and Carlson, 1982; from Alter, 1980)

1. Poor integration of the database with other (internal and external) databases
2. Poor response times
3. Inability to run large models
4. Inability to interface the dialog component with the modeling and database components
5. Inability of maintenance programmers to understand the software structure
6. High development, operating or maintenance costs

Heading off Integration Problems

- Design of DSS must include plan for integrating components
- Make sure you understand how different software packages interface before committing to development environment
- Test out interfaces early during development
- Interface requirements may necessitate suboptimizing on features of components
- Strive for modularity - so that changes in one component do not require redesign of other components

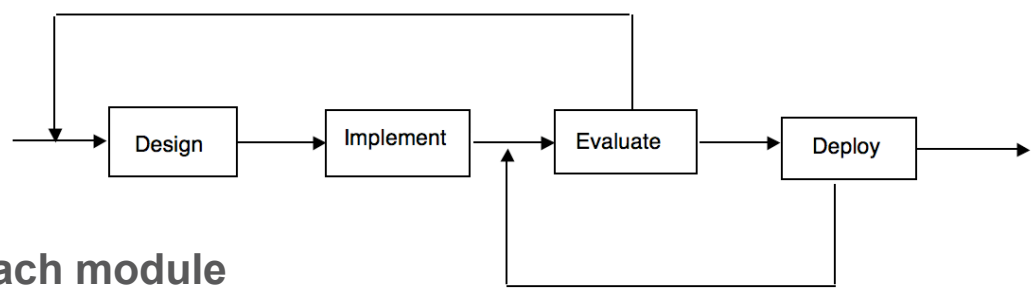
Evolutionary Development

Aspects of Evolution:

- Evolution in response to user feedback
- Evolution toward increasing functionality

Requirements for Evolutionary Development:

- Constant evaluation
- Constant training
- Modular decomposition
- Documentation
  - * existing design&code
    - functionality of each module
    - inputs & outputs of each module
    - relationships between modules
  - * rationale for design decisions
  - * rationale for design changes



Dimensions of Evolution

- Functions are added
- Functions become more easily or naturally accessible to user through dialog subsystem
- Functions become operationally primitive ("black box" to user)
- Subfunctions become accessible to user control (user gets inside "black box")
- New decompositions of existing functions are added
- Changes and additions to exploit new hardware capabilities

Implementing an Evolving System

- **Decompose hierarchically into functional units**
  - Software support: RDD-100, IDEF0, CASE tools
- **Implement functional units as separate modules**
- **Strict control of linkages between modules**
  - data hiding
  - defined dataflows
- **Document extensively**
  - functional decomposition
  - internal logic of each module
  - data sharing: inputs, outputs, shared data structures
- **Maintain strict version control**
- **Evaluate constantly**
  - Test modules separately before interconnecting
  - Test each increment in functionality before adding more
- **Involve users as early and as often as possible**

Managing Evolution

- **Follow sound configuration management practices**
- **Adhere to strict protocol for updating current working version:**
  - thorough testing of changes
  - document rationale for changes
  - protocol for approval & addition of changes
- **Maintain history:**
  - old versions
  - old documentation
  - rationale for changes

Involving Users During Implementation

- Aspects requiring user feedback:
  - user interface
  - methods:
    * input s required
    * outputs available
    * adequacy of methods for task (knowledgeable users)
    * understandability of explanations
  - data:
    * used by system
    * available to user
- User evaluation is part of change testing
- User response should be recorded as part of design documentation

Deploying an Evolving System

- Test before deploying
- Don't deploy too soon
  - all the bugs are worked out
  - improvement is worth learning cost
- Recruit alpha and beta testers
- Solicit (and use!) user feedback
- Provide good documentation of changes
- Provide good training for changes
- Provide accessible user support

Deployment of Evolutionary Changes

- **Establish and adhere to interface conventions**
  - Between modules
  - With outside world
- **Incremental training**
  - Clear written and on-line documentation
  - User on development team should be heavily involved in training
  - Embedded intelligent training
- **User support**
  - Close cooperation between training and user support teams
  - Respect for users is essential!

Interaction with User Community

- Development team should be:
  - Respectful of user competence
  - Genuinely sensitive to user concerns
  - Actively responding to user feedback
  - Willing to admit and correct mistakes
- How you handle the inevitable problems strongly affects acceptance of the final product
- Consider having a "user liason" to monitor and respond to problems

Approaches to DSS Development

1. Off-the-shelf DSS
2. Build specific DSS from DSS generator
3. Build specific DSS from DSS tools
4. Build specific DSS from ground up
5. Use specific DSS development efforts to build and refine DSS generator

DSS Generators

- DSS-building environment / toolkit
  - User interface builder
  - Data entry, storage, query capability
  - Model development tools
  - Runtime application constructor
- Examples
  - Spreadsheets and add-ins, templates, macro and programming languages
  - DBMS
  - Modeling packages (SPSS, IFPS, Matlab)
  - General purpose DSS (Hugin, Analytica, Expert Choice)
- Many organizations that develop DSS create their own generators

DSS Tools

- Graphics packages
- Interface builders
- DBMS
- GIS
- Statistical software libraries
- Management science software libraries
  - optimization
  - decision analysis
  - simulation
- Expert system shells
- Web development toolkits

The boundary between DSS tools and DSS generators is fuzzy

In Summary...

References for Unit 7

Andriole, S., Handbook of Decision Support Systems, TAB Books, Inc., 1989.

Boehm, B.W. A Spiral Model of Software Development and Enhancement. IEEE Computer, 21(5), 61-72, 1988.

Boehm, B.W., Gray, T.E. and Seewaldt, T. Prototyping vs. Specification: A Multi-Project Experiment. Proceedings of Seventh International Conference on Software Engineering, New York: ACM and IEEE, 473-484, 1984.

Buede, D. *The Engineering Design of Systems*: *Models and Methods*. Wiley, 2000.

Jonassen, D., Hannum, W., and Tessmer, M. *Handbook of Task Analysis Procedures*. Praeger, 1989.

McGraw, K. and Harbison, K. *User-Centered Requirement: The Scenario-Based Engineering Process*. Erlbaum, 1997.

Sage, A. Decision Support Systems Engineering, New York: Wiley, 1991.
Young, R. *Effective Requirements Practices*. Addison-Wesley, 2001.