A Decision Support Framework for the Software Systems Engineer

Dr. Bob Knapper, The MITRE Corporation

Dr. Peggy Brouse, George Mason University

Presented by Rosana Stoica

9 July 2012

Agenda

- Introduction
- Problem Statement
- Research Methodology
- Case Study
- Architecture Artifact Restructuring Techniques (AARTs)
- AARTs Validation
- Experiment Findings and Conclusions
- Software Engineering Decision Support
- Conclusions
- Open Research Items
- Questions ?

Introduction

- Legacy Systems
  - Any fielded system is a legacy system
  - Constructed with varying technologies and for varying timeframes of utility
  - Retiring legacy systems is a consideration between the cost to reengineer or replace vs. cost to retain/maintain
- Integrating “as-is” into “to-be”
  - Requirements engineering and “use-case” analysis critical
- Challenges
  - Legacy systems may be poorly documented
  - Requirements may be non-existent
  - Stovepipes may be resistant to integration

Problem Statement

- System-Level Software challenges exist in development, maintenance, reengineering, or modernization activities.
- Legacy systems are often retained beyond their original expected period of performance for many reasons.
- Monolithic (or stovepiped) systems are often considered as targets for renewal and integration into new systems.
- Disparate systems should be viewed at the same abstract architecture level for consistency.
- Techniques, tools, and methodologies employed at a common abstract architecture level for disparate systems is of benefit.

Research Methodology

- Phase I – Performed a Case Study
  - Assessed the use of an architecture framework to create valuable components for further use in the integration lifecycle
  - Developed and documented architecture artifact restructuring techniques (AARTs) for manipulating architecture components
- Phase II – Performed an Experimental Study
  - Designed an experiment to test AARTs against hypotheses
  - Executed the experiment and determined the efficacy of applying AARTs in real-world development efforts
  - Developed conclusions based on findings and analyses
- Documented overall conclusions and identified open research items

Case Study

- Phase I – A Case Study
  - Single case to be studied:
    - Extremely large DoD enterprise architecture (EA) effort was attempted.
    - EA documented the Operational and System architecture views partially for the “as-is” enterprise and the projected “to-be” enterprise.
    - The “to-be” enterprise system comprised legacy systems and new development.
    - Initial version of the Operational and a draft System architecture views were developed and analyzed during the study period.
  - The 12-month Case Study was conducted to gather information, make observations, and to explore “what if” exercises with respect to architecture frameworks and the integration of the architecture components.

Case Study Conclusions

- DoD Architecture Framework provided adequate high-level guidance, but required experienced modelers and subject matter experts to be effective.
- The one-year effort to fully document the Enterprise Architecture generated 100's of architecture artifacts, but was less than successful:
  - Many gaps remained in the limited "as-is" architecture coverage.
  - Integration was incomplete.
  - System View artifacts did not map well to Operational View artifacts.
- Current architecture documentation methods used to generate meta-level artifacts are not capable of predictive analyses for integration on their own.
- Given the nature of the relatively poorly constructed / documented architectures, opportunities abounded for applying new techniques (e.g. AARTs, see next slide).

Architecture Artifact Restructuring Techniques (AARTs)

- A group of architecture artifact restructuring techniques is defined:
  - *Adapterize* - create an adapter between components
  - *Containerize* - employ "wrapping" of components
  - *Assimilate* - subjugate one component to another
  - *Conform* - modify one component and/or interface
  - *Equalize* - modify both components and/or interfaces

Commonly Known and Applied (Adapterize, Containerize)

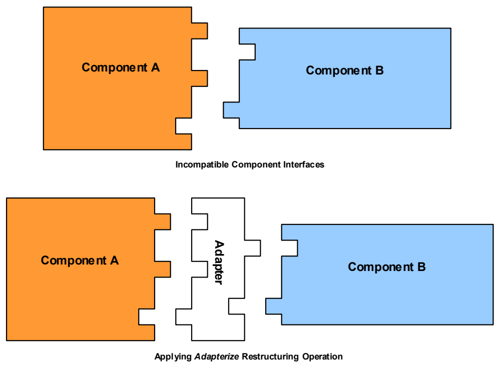
Less Common (Assimilate, Conform, Equalize)

- These techniques are intended for use with abstract representations of architecture components.
- Applying AARTs to architecture components results changes to the architecture which guides and assists applicable changes to the underlying implementation.
- Benefits to using AARTs include reduced complexity, elimination of duplicate functionality, preservation of critical functionality, and ease of future maintenance.

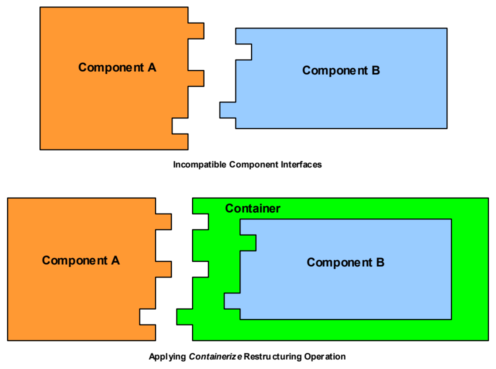
AARTs Graphical Depiction

Commonly Known and Applied

Adapterize

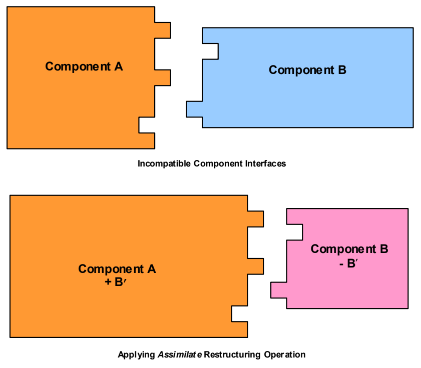

Component A
Component B
Incompatible Component Interfaces
Component A
Adapter
Component B
Applying *Adapterize* Restructuring Operation

Containerize

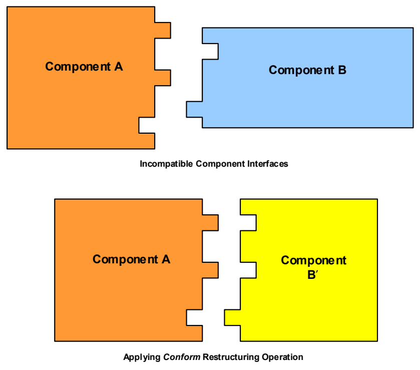

Component A
Component B
Incompatible Component Interfaces
Component A
Container
Component B
Applying *Containerize* Restructuring Operation

Less Common

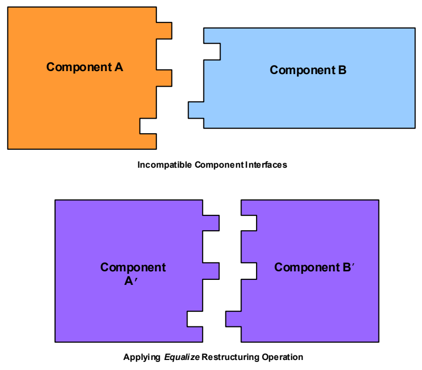
Assimilate


Component A
Component B
Incompatible Component Interfaces
Component A + B′
Component B - B′
Applying *Assimilate* Restructuring Operation

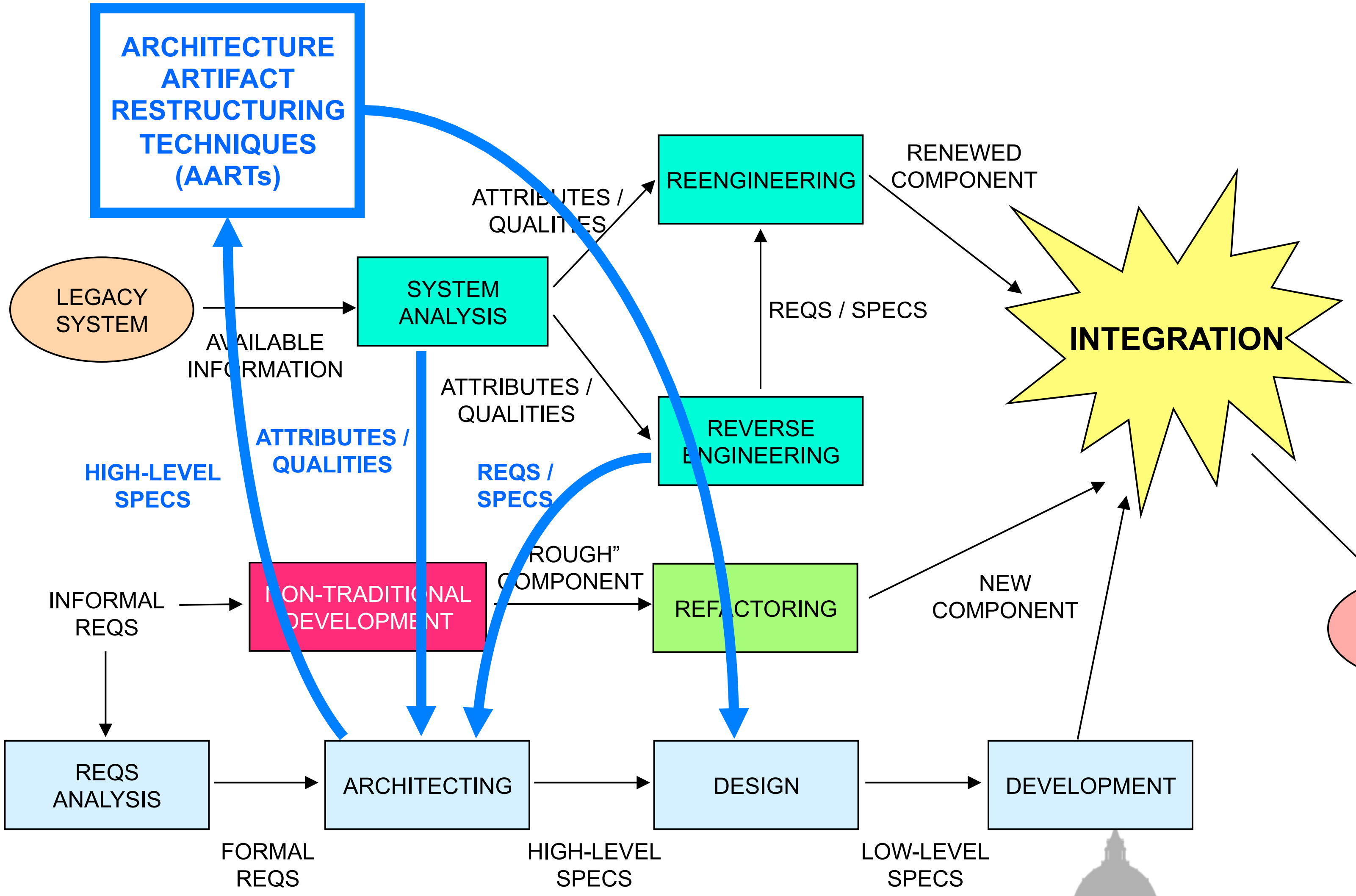
Conform


Component A
Component B
Incompatible Component Interfaces
Component A
Component B′
Applying *Conform* Restructuring Operation

Equalize


Component A
Component B
Incompatible Component Interfaces
Component A′
Component B′
Applying *Equalize* Restructuring Operation

AARTs in the Integration Engineering Lifecycle



Architecture Restructuring Experiment Using AARTs

- Phase II – An Experimental Study
  - The Informational Case Study yielded five notional architecture artifact restructuring techniques.
  - An algorithm to apply those techniques is defined:
- Given

  $T(P_i, P_j, R_k)$ is the restructuring transformation

  P is the set of program components to be integrated

  P_i and **P_j** are elements of **P**

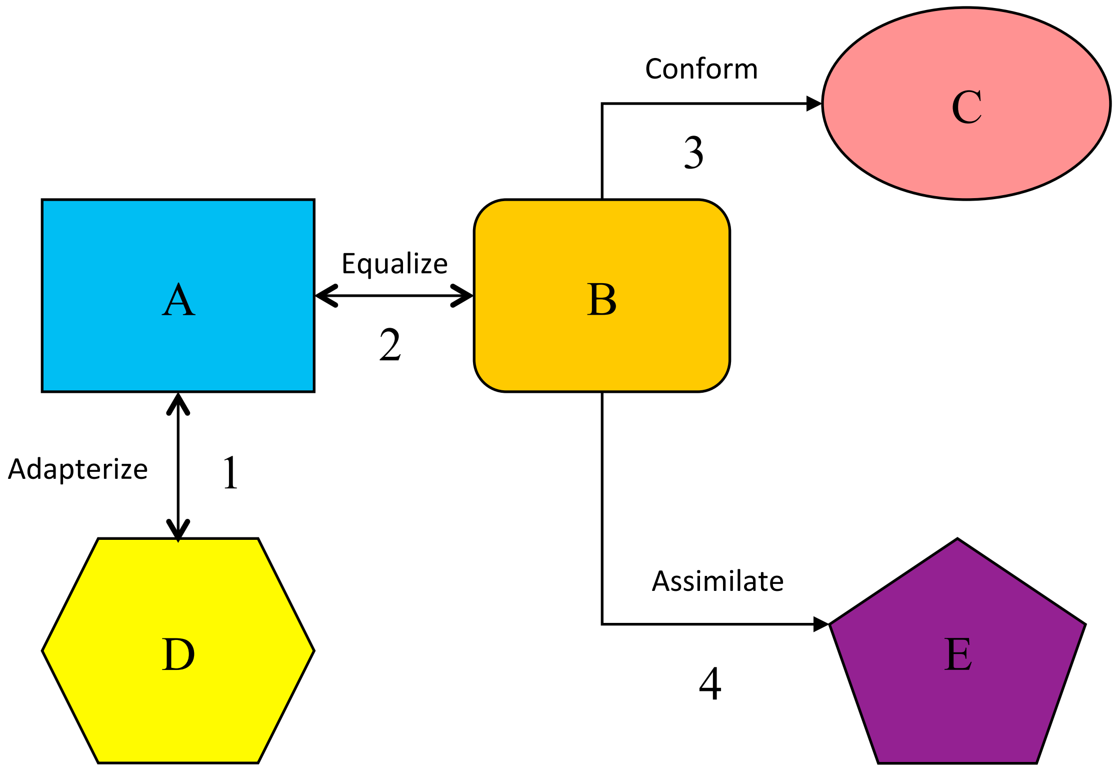
  R is the set of restructuring operations

  R_k is an element of **R**

  I is the Integrated System
- Then

  $I \leftarrow \sum T(P_i, P_j, R_k) \; \forall \; P_i , P_j \in P \wedge R_k \in R$
- Adapterize and Containerize are state-of-the-practice and will be designated as the control group in the experimental study.

Restructuring Example



if   P = (A, B, C, D, E) and R = (Adapterize, Equalize, Conform, Assimilate) then
I = T(A, D, Adapterize) + T(A, B, Equalize) + T(B, C, Conform) + T(B, E, Assimilate)

AARTs Hypotheses to be Validated

- H1: Using the Adapterize and Containerize restructuring techniques yield an integration outcome with higher cost to using the Assimilate, Conform, and Equalize restructuring techniques.
- H2: Using the Assimilate, Conform, and Equalize restructuring techniques yield a reduction in size growth compared to using the Adapterize and Containerize restructuring techniques only.
- H3: Using the Assimilate, Conform, and Equalize restructuring techniques yield an integration outcome with a reduction in cost compared to using the Adapterize and Containerize restructuring techniques only.
- H4: Using the Assimilate, Conform, and Equalize restructuring techniques yield an integration outcome with a reduction in complexity compared to using the Adapterize and Containerize restructuring techniques only.

AARTs Experiment

- Program Objective
  - Develop an armored, self-protecting, personnel carrier, capable of multi-terrain operations
  - Vehicle is to be software-controlled to a maximum extent
  - Minimize new software development and use applicable legacy systems as needed to achieve this
- Developers (three experienced CMMI Level 5 DoD software providers, biases assessed and mitigated)
  - Labeled Developer A, Developer B and Developer C
- Subsystems Architecture Artifacts available for13 subsystems
  - Labeled Subsystem A through Subsystem M for the purposes of this presentation.

Experiment Observations

- Developer A
  - Took the most conservative approach to the integration effort, developing adapters and wrappers rather than the more invasive techniques requiring modification of the existing subsystems.
  - Developer B and Developer C were compared against Developer A (since they used the "control" techniques only).
- Developer B
  - Had a less conservative set of restructurings. This architecture changed considerably over Developer A since the restructurings involve modifying interfaces and in some cases the subsystems functionality.
- Developer C
  - Like Developer B they made modifications to one or more subsystem interfaces and in some cases re-architected parts of the subsystem to better facilitate integration. The choices they made to modify rather than just wrap were based on creating systems that were seamless in their integrated design structure in order to enhance the systems maintainability for the future.

Experiment Analysis

- Analyzing the data involved computing the integrating value from each developer along with the value of each method and its significance with respect to the hypotheses. In addition, a "decision value" is computed for use by a decision support framework.
- The values shown in the subsequent summary tables are defined as follows:
  - I – percentage of change expected in subsystem size
  - C – ratio of "plug and play" cost to expected cost
  - M – quartile rank of expected subsystem McCabe complexity
  - D – decision value (lower is better)
- The values I, C and M are normalized to be in the 1-100 range yielding I', C' and M' and are weighted x, y and z respectively. The decision value D is then computed as: $D = x(I') + y(C') + z(M')$ to yield a result between 1 and 100.

Experiment Analysis (concluded)

- A survey of 102 software and system engineering professionals was conducted to determine what values for x, y, and z should be chosen to provide a “sensitivity” validation of the data.
- The majority of the respondents (all of whom support the U.S. Government on various programs) indicated that cost reduction was their principal factor and they weighted it heavily (75% - 90%).
- There were also responses favoring an equal weighting as well as those favoring small increases in size (10% - 20%).
- None of the respondents favored weighting the complexity attribute heavily.
- To assist in the analysis, the responses were grouped into six clusters of similar values for x, y, and z. The clusters are: (0.1, 0.8, 0.1), (0.3, 0.5, 0.2), (0.333, 0.334, 0.333), (0.4, 0.5, 0.1), (0.5, 0.4, 0.1), and (0.8, 0.1, 0.1). Values for D were computed on these clusters.
- Additional ranges for x, y, and z were computed zeroing out one attribute and varying the others in increments of 0.1.

Experiment Results

Developer A used only Adapters and Wrappers

Data for Developer A with Attributes from Experts

Subsystem	Restructuring Techniques				I=0.1, C=0.8, M=0.1	I=0.3, C=0.5, M=0.2	I=0.333, C=0.334, M=0.333	I=0.4, C=0.5, M=0.1	I=0.5, C=0.4, M=0.1	I=0.8, C=0.1, M=0.1
		I'	C'	M'	D	D	D	D	D	D
Subsystem A	Adapterize	11	14	2	12.5	10.7	9.005	11.6	11.3	10.4
Subsystem B	Adapterize	14	19	2	16.8	14.1	11.674	15.3	14.8	13.3
Subsystem C	Containerize	17	29	3	25.2	20.2	16.346	21.6	20.4	16.8
Subsystem D	Adapterize	9	13	2	11.5	9.6	8.005	10.3	9.9	8.7
Subsystem E	Containerize	11	14	2	12.5	10.7	9.005	11.6	11.3	10.4
Subsystem F	Containerize	13	18	2	15.9	13.3	11.007	14.4	13.9	12.4
Subsystem G	Containerize	9	11	2	9.9	8.6	7.337	9.3	9.1	8.5
Subsystem H	Containerize	9	11	2	9.9	8.6	7.337	9.3	9.1	8.5
Subsystem I	Containerize	23	52	3	44.2	33.5	26.026	35.5	32.6	23.9
Subsystem J	Containerize	13	17	2	15.1	12.8	10.673	13.9	13.5	12.3
Subsystem K	Containerize	19	37	3	31.8	24.8	19.684	26.4	24.6	19.2
Subsystem L	Adapterize	10	12	2	10.8	9.4	8.004	10.2	10	9.4
Subsystem M	Containerize	9	12	2	10.7	9.1	7.671	9.8	9.5	8.6
	Average	12.85	19.92	2.23	17.45	14.26	11.675	15.32	14.62	12.49

Experiment Results (continued)

Developer B used Conform, Assimilate, and Equalize only

Data for Developer B with Attributes from Experts

Subsystem	Restructuring Techniques				I=0.1, C=0.8, M=0.1	I=0.3, C=0.5, M=0.2	I=0.333, C=0.334, M=0.333	I=0.4, C=0.5, M=0.1	I=0.5, C=0.4, M=0.1	I=0.8, C=0.1, M=0.1
		I'	C'	M'	D	D	D	D	D	D
Subsystem A	Conform	3	5	2	4.5	3.8	3.335	3.9	3.7	3.1
Subsystem B	Conform	4	7	2	6.2	5.1	4.336	5.3	5	4.1
Subsystem C	Assimilate	11	15	3	13.4	11.4	9.672	12.2	11.8	10.6
Subsystem D	Conform	3	5	2	4.5	3.8	3.335	3.9	3.7	3.1
Subsystem E	Assimilate	9	13	2	11.5	9.6	8.005	10.3	9.9	8.7
Subsystem F	Assimilate	9	12	2	10.7	9.1	7.671	9.8	9.5	8.6
Subsystem G	Assimilate	5	7	2	6.3	5.4	4.669	5.7	5.5	4.9
Subsystem H	Equalize	7	11	2	9.7	8	6.671	8.5	8.1	6.9
Subsystem I	Equalize	11	14	3	12.6	10.9	9.338	11.7	11.4	10.5
Subsystem J	Equalize	6	9	2	8	6.7	5.67	7.1	6.8	5.9
Subsystem K	Assimilate	12	18	3	15.9	13.2	11.007	14.1	13.5	11.7
Subsystem L	Conform	3	5	2	4.5	3.8	3.335	3.9	3.7	3.1
Subsystem M	Equalize	5	7	2	6.3	5.4	4.669	5.7	5.5	4.9
	Average	6.77	9.85	2.23	8.78	7.4	6.286	7.85	7.55	6.62

Computed values all less than Developer A's

Experiment Results (concluded)

Developer C used Conform, Assimilate, and Equalize only

Data for Developer C with Attributes from Experts

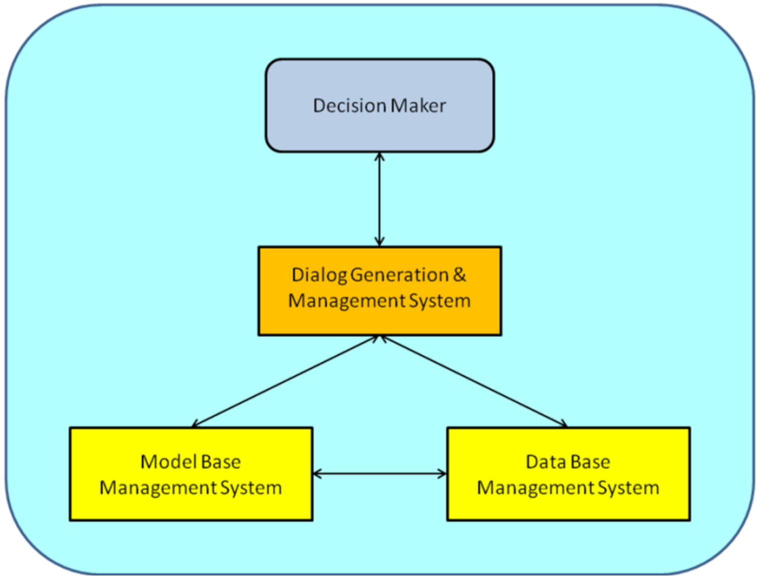
Subsystem	Restructuring Techniques				I=0.1, C=0.8, M=0.1	I=0.3, C=0.5, M=0.2	I=0.333, C=0.334, M=0.333	I=0.4, C=0.5, M=0.1	I=0.5, C=0.4, M=0.1	I=0.8, C=0.1, M=0.1
		I'	C'	M'	D	D	D	D	D	D
Subsystem A	Conform	3	5	2	4.5	3.8	3.335	3.9	3.7	3.1
Subsystem B	Assimilate	5	7	2	6.3	5.4	4.669	5.7	5.5	4.9
Subsystem C	Equalize	11	14	3	12.6	10.9	9.338	11.7	11.4	10.5
Subsystem D	Conform	3	5	2	4.5	3.8	3.335	3.9	3.7	3.1
Subsystem E	Conform	3	5	2	4.5	3.8	3.335	3.9	3.7	3.1
Subsystem F	Equalize	5	7	2	6.3	5.4	4.669	5.7	5.5	4.9
Subsystem G	Assimilate	7	11	2	9.7	8	6.671	8.5	8.1	6.9
Subsystem H	Assimilate	5	8	2	7.1	5.9	5.003	6.2	5.9	5
Subsystem I	Equalize	9	12	3	10.8	9.3	8.004	9.9	9.6	8.7
Subsystem J	Assimilate	5	7	2	6.3	5.4	4.669	5.7	5.5	4.9
Subsystem K	Equalize	13	16	3	14.4	12.5	10.672	13.5	13.2	12.3
Subsystem L	Conform	3	5	2	4.5	3.8	3.335	3.9	3.7	3.1
Subsystem M	Assimilate	5	7	2	6.3	5.4	4.669	5.7	5.5	4.9
	Average	5.92	8.38	2.23	7.52	6.42	5.516	6.78	6.54	5.8

Computed values all less than Developer A's

Experiment Findings and Conclusions

- AARTs Hypotheses - restated
  - H1: Using the Adapterize and Containerize restructuring techniques yield an integration outcome with higher cost to using the Assimilate, Conform, and Equalize restructuring techniques.
  - H2: Using the Assimilate, Conform, and Equalize restructuring techniques yield a reduction in size growth compared to using the Adapterize and Containerize restructuring techniques only.
  - H3: Using the Assimilate, Conform, and Equalize restructuring techniques yield an integration outcome with a reduction in cost compared to using the Adapterize and Containerize restructuring techniques only.
  - H4: Using the Assimilate, Conform, and Equalize restructuring techniques yield an integration outcome with a reduction in complexity compared to using the Adapterize and Containerize restructuring techniques only.
- The assessment of the validity of the four stated hypotheses is as follows:
  - **H1: Validated.** Using Adapterize and Containerize yielded noticeably higher cost than Assimilate, Conform and Equalize.
  - **H2: Validated.** Using Assimilate, Conform and Equalize yielded a reduction in size growth over using Adapterize and Containerize only.
  - **H3: Validated.** Using Assimilate, Conform and Equalize yielded a reduction in cost over using Adapterize and Containerize only.
  - **H4: Not validated.** Using Assimilate, Conform and Equalize did not yield a reduction in complexity over using Adapterize and Containerize only.

Leveraging the Study for Decision Support


Decision Maker
Dialog Generation & Management System
Model Base Management System
Data Base Management System

Notional Decision Support System

- Decision Maker (i.e. the user)
- Dialog Generation & Management System (i.e. the user interface)
- Data Base Management System (i.e. the data store)
- Model Base Management System (i.e. the "engine" that creates the alternatives)

Software Engineering Decision Making

- A decision support system framework can be constructed using the information developed and derived from our research
  - The user interface would provide inputs for the Software Systems Engineer/Architect decision maker to have control over variables such as cost, schedule and performance
  - Our experiential data would be used to populate the data store and to contribute to heuristics in the model base engine
- Given different emphasis on the variables taken individually and as a whole would yield different outcomes for the decision maker
  - E.g. cost-only, schedule-only, or cost/schedule/performance as a specific combination
- Based on the priorities of the overall system:
  - Multiple alternative paths may be prototyped
  - The best alternative may be selected only

Conclusions

- Legacy systems are often retained beyond their original expected period of performance for many reasons.
- Legacy systems with extended lives are targets for renewal and integration into modern systems and pose challenges.
- Viewing legacy systems components and modern/new development components at the same abstract architecture level helps to “level the playing field” of integrating subsystems into systems.
- The research presented herein showed the development and validation of AARTs for use in aiding the Software System Engineer/ Software Architect.
- Development of a Decision Support System interface would allow a user to explore various combinations of restructurings against a set of subsystem architectures to create a range of alternatives.

Contact Information

Dr. Robert J. Knapper
The MITRE Corporation
925 Corporate Drive, Suite 301
Stafford, VA 22554
+1-703-445-3247
rknapper@mitre.org

Dr. Peggy S. Brouse
George Mason University
2215 Nguyen Engineering Building
Fairfax, VA 22030
+1-703-993-1502
pbrouse@gmu.edu

Questions?