

An Enterprise Architecture Practitioner's Notes

VOLUME 3: SOLUTION LEVEL ARCHITECTURE

Matthew Ford Kern

Copyright 2014 to 2015

\$200.00

ISBN 978-1-4951-7313-4

20000 >



An Enterprise Architecture Practitioner's Notes: Volume 3 Solution Level Architecture

Nature of Release	Changes Made	Author Initials	Date
Sent to Editing	Initial Draft	MFK	28 September 2015
GBG Edits	Various suggestions	PV & DC	8 October 2015
More Edits	Various	MFK	25 October 2015
Edition 1	PDF	MFK	25 October 2015

Title: An Enterprise Architecture Practitioner's Notes

Subtitle: Volume 3: Solution Level Architecture

Self-Published, 2015

Matthew Ford Kern, MsEA BsEE CEA(DODAF, FEAF and Zachman), CISSP-ISSAP PMP

ISBN 978-1-4951-7313-4

Price for spiral bound copy, or commercial purposes: \$200

Use as an eBook free for education and government.

CONTENTS

Volume 3: Solution Level Architecture.....5

Section 1: Introduction to Volume 3.....6

1.1 My Ideas: Mostly Not, May 25, 20157

1.2 The Five Activities of Enterprise Architecture, Apr 25, 2015.....9

1.3 Enterprise, Segment, Solution, Jan 14, 2015.....11

1.4 Correspondence to Management Activities, Never Posted.....15

1.5 Strategic, Operational and Tactical Thinking, Jul 19, 2014.....18

1.6 Why Do Architecture?, Jul 5, 201520

1.7 Introducing Architecture to your Organization, June 7, 2015.....23

1.8 EA, BA, DA, Carts and Horses, JUNE 28, 201527

1.9 Achieving Success, April 12, 2015.....28

Section 2: Solution Level Architecture:30

2.2 Integration, System Engineering & Enterprise Architecture, January 11, 2015.....33

2.3 What is System Integration?, August 23, 2014.....35

2.4 Enterprise Architecture vs Systems Engineering, August 16, 2014.....38

2.5 Three Views of System Engineering, January 10, 201541

2.6 Enterprise System Requirements, December 13, 2014.....42

2.7 Allocating Requirements, July 12, 2014.....46

2.8 Basic System Test & Evaluation, March 24, 201548

2.9 Choosing Components, Sep 1, 2015.....50

2.10 Drawing the Box, June 22, 201451

2.11 Drawing Boxes, December 19, 201453

2.12 Managing Interfaces, February 13, 2015.....55

2.13 Functional, Virtual and Physical Architecture, Sep 1, 2015.....57

2.14 System Architecture Quality, February 21, 2015.....59

2.15 Website Solution Methodology, March 22, 2015.....61

2.16 About Enterprise Software, December 24, 2014.....63

2.17 The Value of Enterprise Software, August 3, 2014.....66

2.18 Centralization vs Decentralization, Aug 19, 2015.....68

2.19 The Very Largest Distributed Systems, September 14, 201473

Section 3: Topics:.....75

3.1 Why Federal IT Projects Fail, Sep 16, 2015.....76

3.2 COTS/GOTS vs MOTS vs Custom Development, July 4, 201478

3.3 The Albino Rhino Hunt, June 13, 2015	81
3.4 Is the Cloud New, June 29, 2015	83
3.5 Cloud Service Categories, August 23, 201	85
3.6 The Cloud Paradox, SEPTEMBER 28, 2014	87
3.7 Security vs Cloud, August 23, 2014	89
3.8 Essential Cloud Architecture for Customers, March 21, 2015	91
3.9 The Cloud Killer, June 30, 2015	96
3.10 What is Big Data & Why No-SQL?, August 23, 2014	99
3.11 Basic Truth: OLTP and OLAP, June 1, 2014	103
3.12 Fun with Database Servers, Sep 4, 2015	105
3.13 Building Large Transactional Application Suites, July 15, 2014	108
3.13 Solution Architecture Framework, Never Posted	111
3.14 System Development Lifecycle, Never Posted	114
Section 4: Examples:	116
3.1 A Nationwide Sensor System, December 26, 2014	117
4.2 A Global Nuclear Detection System, January 11, 2015	119
4.3 All Hazards vs Disaster Response, March 22, 2015	122
4.4 Geospatial Routing: EDXL-DE & CoT, June 30, 2015	127
4.5 EDXL is NOT ENOUGH, June 6, 2015	129
4.6 Mobile Network Architecture, Jul 28, 2015	131
4.7 IoT and Web Services	134
4.8 Building a Global Grid for IoT, June 27, 2015	136
4.9 Asynchronous and Synchronous Messages in the IoT, June 27, 2015	138
4.10 IoT Device Classes	140
4.11 Cyberspace	144
4.12 A Legacy System Upgrade, December 24, 2014	146
4.13 Architecture Lessons of the CHAPS Failure, October 20, 2014	148
4.14 Defeating DDOS, December 24, 2014	151
4.15 Advanced Security from the Trash Bin, October 15, 2014	154
4.15 Crypto and the Multiplexer Array, June 17, 2015	156
4.17 A Missing Unix Network Service, June 17, 2015	158
4.18 Another Access Control Model, June 6, 2015	161
4.19 Security Challenge-Response Example, JUNE 25, 2015	163
4.20 How to do Architecture: X-Browser, July 2, 2015	164
Ending Remarks for Volume 3	167

VOLUME 3: SOLUTION LEVEL ARCHITECTURE

This series of books will describe what I view as the totality of enterprise architecture, in the broad meaning of that term. It will consist of five volumes as follows:

1. Enterprise Level Architecture
2. Segment Level Architecture
3. Solution Level Architecture
4. Governance and Maturity Management
5. The Business Environment

While the series will address enterprise architecture (EA) in the broad sense, this volume will address enterprise level architecture in the narrow sense. It is this narrow activity that gives the whole its name, purpose and organizing principles.

These volumes originated as blog posts on my personal website, and later on LinkedIn™. I wrote these to popularize and clarify the deep understanding of enterprise architecture of my government colleagues, who invented it. It seems to me that knowledge of the subject has spread from its global hub city, Washington DC, out to the rest of the globe. We know it best here.

As for myself, my main contribution at the enterprise level is simply popularizing, restating the work of the various founders of EA, of whom I know many. Some are friends; all are colleagues. They created a comprehensive vision of EA, and I follow it. This is mostly true of EA maturity management and EA governance as well. In regard to solution architecture and segment architecture, I have contributed some innovations.

I did contribute the organizational structure for the book, the “Five Activities Model.” It is a small and obvious addition to the work of Dick Burk while at the Office of Management and Budget (OMB).

This work is not another framework and is not intended to replace them. It is designed to, instead, provide what they do not: perspective. This particular volume draws mainly on Federal Enterprise Architecture Framework (FEAF) and FEA concepts, with Zachman's concepts, as those are more directly applicable to this level of architecture and scope of effort.

Each day I hope to do something useful. It is not an ambitious philosophy, but it helps in consulting. With this book I also hope that I have done something useful. Please let me know if I have.

SECTION 1: INTRODUCTION TO VOLUME 3

Volume 3: Solution Level Architecture	5
Section 1: Introduction to Volume 3	6
1.1 My Ideas: Mostly Not, May 25, 2015	7
1.2 The Five Activities of Enterprise Architecture, Apr 25, 2015	9
1.3 Enterprise, Segment, Solution, Jan 14, 2015	11
1.4 Correspondence to Management Activities, Never Posted	15
1.5 Strategic, Operational and Tactical Thinking, Jul 19, 2014	18
1.6 Why Do Architecture?, Jul 5, 2015	20
1.7 Introducing Architecture to your Organization, June 7, 2015	23
1.8 EA, BA, DA, Carts and Horses, JUNE 28, 2015	27
1.9 Achieving Success, April 12, 2015	28

The section “My Ideas: Mostly Not” describes the limits of my contribution to the art relative to the founders. The other four sections describe the underlying model on which the five volumes are based.

In “The Five Activities...” the basic notion of dividing EA into these five bins is described and then in “Enterprise, Segment, Solution” the preceding model of OMB and Burk is recapped and simplified. In “Correspondence to Management Activities” the link of each of the OMB levels to the PMI levels of management is described. Lastly in “Strategic, Operational and Tactical Thinking” the correspondence to levels of planning is implied.

New material was added for this volume. In “Why Do Architecture” I address the attitude that architecture is not required. In “Introducing Architecture...” I give some notes on initial implementation of an architecture program. In “...Carts and Horses” I address the phenomenon where some organizations promote parts above the whole. In “Achieving Success” I suggest that architecture is used to achieve some goal or goals.

QUESTIONS FOR SECTION 1

1. Do these five levels cover all of enterprise architecture? If not, what is left out?
2. Does the three-level model correspond to contracts and “statements of work” you have seen?
3. How often is it true that the segment architecture works for the program manager? Should it happen more often in an ideal situation? What of the other levels?
4. Have you seen other authors use a different order of operational and tactical planning? Have you seen them refer to all planning as strategic? Is that useful?
5. Why do architecture, in your own words?
6. Is it possible to have architecture without a goal or goals?

1.1 MY IDEAS: MOSTLY NOT, MAY 25, 2015



Several people suggested I should write a book, notably Dale Chalfant, in Detroit (a fine architect), who convinced me. I had toyed with and resisted the idea for several years, as most of what I have learned has come from those founders of enterprise architecture and systems engineering and whomever else I have read. I have added little to the body of thought, a bit here or there, but nothing like their sweeping insights.

You could say that mostly I simply popularize (simplify, clarify and restate) what my government and beltway friends created and regarding enterprise level architecture that would be fair. I think in solution architecture I may have innovated more.

I really don't have that many ideas of my own regarding EA, and of those I have, only a few are profound.

I would like to list some of the folks whose ideas I have borrowed, restated and maybe extended a small bit. They are the real source of what I say. They are not in order of precedence, just random order as I thought of them.

- John A. Zachman, whose ontology I use to think about architecture
- Kathie Sowell, the "mother" of DODAF, who ran the project at MITRE
- Mike Tieman, who rewrote FEAF 1.1 and whose thinking I admire greatly

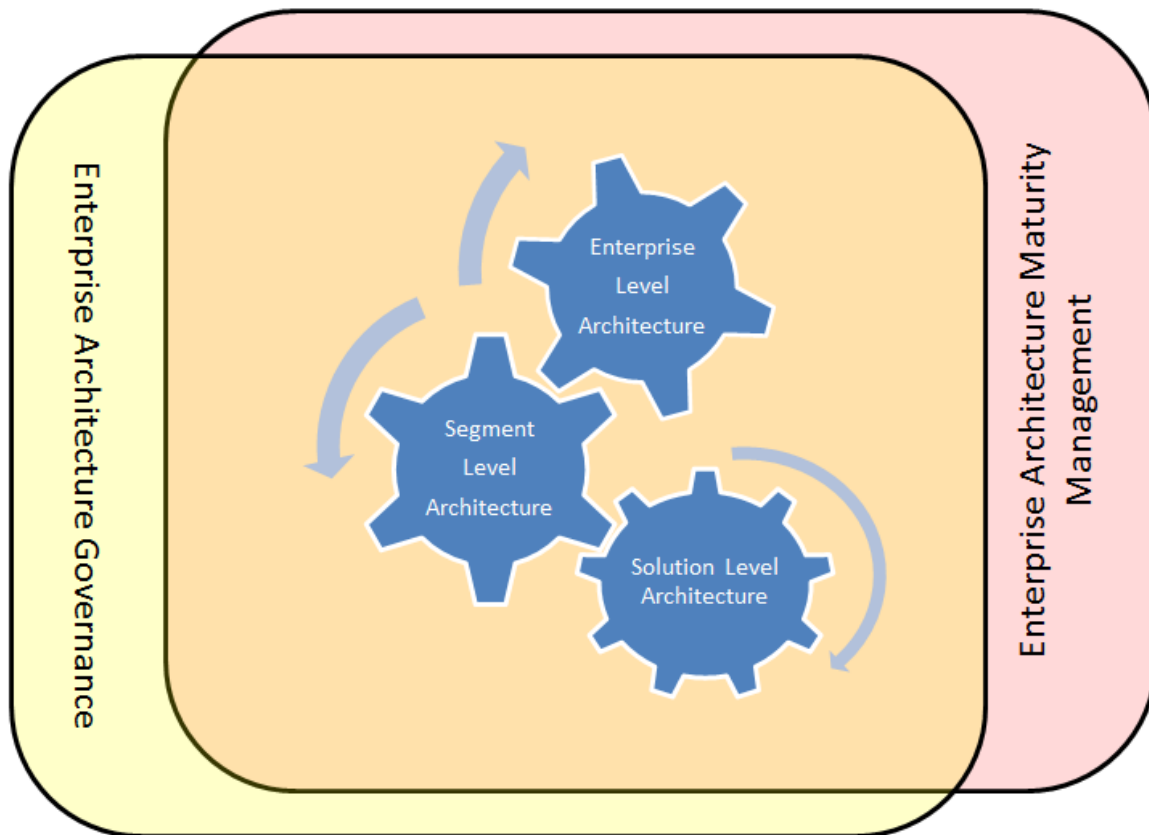
An Enterprise Architecture Practitioner's Notes: Volume 3 Solution Level Architecture

- Felix Rausch and Beryl Bellman who made it possible for me to get an education in enterprise architecture
- Lee Smith, the first Chief Architect at DHS and lion tamer, from whom I learned many lessons concerning governance
- Rob Thomas, who wrote so much of the early material for FEA and FEAF. I met him recently, a great man.
- Manny DeVera, contributor to FEAF 1.1 and a great guy. I worked with Manny a bit at the Federal Emergency Management Agency (FEMA).
- Ira Grossman, who popularized and supported EA for years. I met him on EDMICS and worked with him a bit at FEMA
- Dr. Scott Bernard, a brilliant guy. He was the brains behind FEAF II. Currently Chief Architect at OMB.
- Bobby Jones, who can sell enterprise architecture like no one else. I worked for Bobby at FEMA for a bit.
- Stephen Spewak, who died early and who I never met. I regret that.
- Bradford Rigdon, who chaired the panel for NIST and whose team first used the term "enterprise architecture" defining it by context. I never met him either.
- Richard Burk, former Chief Architect at OMB and a great guy. I quote him often.
- Kshemendra Paul, another former Chief Architect at OMB. I sat in class with him at FEAC, a brilliant guy.
- Randy Hite, who worked tirelessly at the US Government Accountability Office (GAO) for so many years and wrote the Enterprise Architecture Management Maturity Model (EAMMF), a monument to good government.
- Skip Boeteger, my sounding board and more senior colleague. We share neurons, I think.
- I have to add John Tieso. I just saw him at the Business Process Management (BPM) conference, which reminded me. I sometimes forget he is a great architect, as he pretty much agrees with me on nearly everything. I suppose, ironically, that if I had a bigger ego I would remember John is a great architect more often.

It strikes me, having written the list, how many of these folks are among my friends. Also, of those remaining, how many I wish I knew better. Surely I left some out, and I will have to edit them in. (If I did leave you out, it was probably simply my brain misfiring.)

Regardless, they thought the profound thoughts and I followed. I hope I was a good student. If you like what I have said, seek these folks out. I learned from them.

1.2 THE FIVE ACTIVITIES OF ENTERPRISE ARCHITECTURE, APR 25, 2015



Enterprise Architecture is pretty darned simple if you have a good model to explain it. Without a good model everyone starts arguing. In 1989 the Federal Enterprise Architecture Framework (FEAF) described a three-layer model that was pretty simple. In 2006 Burk at OMB described it even more clearly. Three layers—it's easy.

People talked about two other important activities, governance and process improvement of EA. I wrote a paper and added those a few years ago. Five activities—it's simple.

So for just a moment ignore those with partial views, axes to grind or strange garage-grown frameworks and let me explain the five simple activities of a complete enterprise architecture effort.

ENTERPRISE LEVEL ARCHITECTURE

The term enterprise refers to either a whole organization or some hard effort. Here we refer to the whole organization. This has nothing to do with coding information systems or other details, and everything to do with keeping an inventory of all the important features in the enterprise to be transformed and updating this based on plans. It also involves keeping a schedule (roadmap, transition plan) for efforts to change the enterprise. This level supports the portfolio management

efforts encompassing all transformational investments (programs, projects) in the enterprise—all of them.

SEGMENT LEVEL ARCHITECTURE

A "Line of Business" can be a product line or a line of services or some mix. A "Segment" is either one of those or some other large internal effort used across a wide range of lines. The segment architecture describes things like the customers (or stakeholders), the value chain, the logistics chain, the distribution chain, the production line, etc. These usually correspond to a "Program" and so segment architecture usually supports "Program Management." Most importantly, this activity must propose the business cases for the improvements to be funded in the portfolio and implemented in real projects.

SOLUTION LEVEL ARCHITECTURE

The solution is some system created to effect transformation. In changing the organization or the line of business, something must often be automated, centralized, decentralized, constructed, moved or otherwise revamped. The solution architecture describes how that is built, moved, changed etc. Each such thing is a system and a project to be completed.

ENTERPRISE ARCHITECTURE GOVERNANCE

To make all this work you must have a governance structure to tie architecture with real implementation. Otherwise all the stray cats go their own way. You need at least three levels: to approve the portfolio decisions, to approve the business cases to change the segments, and to approve the changes in the solutions (systems).

ENTERPRISE ARCHITECTURE MATURITY MANAGEMENT

All these things are processes. The organization needs some means to manage, standardize and improve these processes.

CONCLUSION

The Five Activities Model is a simple way to understand enterprise architecture.

1.3 ENTERPRISE, SEGMENT, SOLUTION, JAN 14, 2015



FIGURE 1 THREE LEVELS OF ARCHITECTURE FROM 2007 FEA PRACTICE GUIDANCE OF US GOVERNMENT OMB

This section will describe the basic differences in the three levels of architecture presented first in early material on the FEAF and FEA. The three levels were again described by Burk in the 2007 and 2008 FEA Practice Guidance. This model is extremely important in differentiating the types of work in architecture and minimizing redundancy of effort.

The descriptions here are based in part on my own understanding of architecture and experience. For other views, you might check FEA practice guidance, the early FEA documents on establishing enterprise architecture and FEAF v1.1.

I find that this old material is poorly understood outside DC. Even in DC, some practitioners have an inadequate understanding due to lack of education or training. Consequences of mashing the levels together with fuzzy thought processes include less effective architecture, reduced cost effectiveness, poor clarity, redundancy and excess work. Therefore, I find this material important to all practitioners.

ENTERPRISE LEVEL

As shown in the accompanying image, the enterprise level of architecture is intended to be shallow but broad. The intended scope is the entire enterprise—the complete agency, department, corporation or whatever you are charged with. It is focused on strategic outcomes based on strategic planning. The process of ensuring investments and architecture support strategy is called alignment.

The enterprise level of architecture supports the choice of transformation investments. In the Project Management Body of Knowledge (PMBOK) this activity is described as portfolio management, and in the US Government it is called Capital Planning and Investment Control (CPIC). Transformation expenditures in EA are treated as investments and are expected to produce a

return on investment (ROI). Comparative management of investments ensures high ROI¹ and controlled risk.

To minimize unneeded depth (detail) and maximize utility, simple inventories of the major elements of the enterprise are kept at the enterprise level (composition). Relationships between elements of the inventories are kept (structure) to understand the effects of change. By comparing the current enterprise to the target enterprise (the composition and structure after investments are applied), you can determine remaining gaps.

Other than inventory lists and their relationships, the main artifact at the enterprise level is the transition plan or roadmap, a schedule of initiation and completion of each investment leading to the target state. (One good practice is to include stage gates, color coded for systems development life cycle (SDLC) stages and initial operating capability/full operating capability (IOC/FOC) on the investment lines of this schedule.) Artifacts should not include the types listed for lower levels of architecture, as these would be redundant, unless a clear need exists.

Vision, standards, principles and other guidance are commonly produced at this level for consumption by the levels below.

For more on a minimalist approach to the enterprise level see:
<https://www.linkedin.com/pulse/20140727145732-86002769-very-lean-enterprise-architecture?trk=mp-reader-card>

SEGMENT LEVEL

The segment level of architecture is less broad and more detailed than the enterprise level. It is also wider and less deep than solution architecture. The segment level is focused on the operational mission and on operational plans. One primary purpose of this level is to produce the business plans that propose new transformation investments (to be reviewed and selected at higher levels). The segment level also introduces customer focus and ensures individual systems add value to the operations.

The segment level describes lines of business. This would include the products that compose a product line or the services that compose a line of services. It would also include the mix of products and services in a line of business.

The governance body that most often appears at this level is that which makes stage-gate review decisions, which oversees lower systems engineering and subsumed solution architecture. Some SDLC context is often applied.

Coverage might include the supply chain, the manufacturing line, the value chain, the distribution chain, markets and customers. Segment architecture is best when focused on the value delivered to the customer, or in government, the value delivered to the citizen.

¹ ROI is often measured in terms on intangible returns in government, such as improvements to organizational performance. This occurs because there is no revenue, no profit, at that organization.

Various operational diagrams are the main artifacts of segment architecture. The value chain diagram is of particular note. The key artifacts do not include redundant listed inventories of what exists in the enterprise nor roadmaps.

Three kinds of segments are often described. The first includes all the mission segments, a.k.a. the core business of the organization. The second category includes all support operations, such as human resources. The third includes any internal initiatives to provide a common resource to the organization, such as an Enterprise Service Bus (ESB) in IT, or a fleet of cars and trucks for non-IT. Each may have many component solutions implemented as projects, so the segment level can be said to correspond to the program level in the PMBOK.

For more on a customer focused approach to Segment Architecture see:

<https://www.linkedin.com/pulse/20140727163249-86002769-customer-centric-enterprise-architecture?trk=mp-reader-card>

SOLUTION LEVEL

The solution level of architecture describes the particular detailed implementation plans of one project or investment. Often this may be produced by the contracted company implementing the plan, unlike the levels above. Only at this level is discussion of servers, virtual servers, programming languages and features (Struts, Hibernate, etc....) is appropriate.

The governance body most often associated with this level of architecture is the CCB (configuration control board). Decisions supported by this level of architecture are considered tactical in the enterprise context.

A wide range of artifacts are possible at the solution level describing operations, business process, databases, software structure or service-oriented architecture (SOA) services, component applications, ESBs, and other such details of implementation. These artifacts should not be redundantly reproduced at higher levels. Higher level artifacts are commonly referenced.

Any solution exists within the context of improvement of a segment.

INTEGRATION

In the enterprise repository the segment artifacts are commonly attached to the enterprise. The solution artifacts are attached to the segment in the EA repository as well. Segment artifacts are filed and kept together by some mechanism in the repository, as are solutions. Repository tools such as Trough Architect™ are designed to do precisely these things.

In a medium or large enterprise, different teams may produce different instances at different levels of architecture. The enterprise level is most often reserved for the employees of the enterprise. This solution level is often contracted out with the solution, producing innovation and other advantages. Guidance on these different levels helps to streamline these distributed efforts.

TERMINOLOGY

In a loose sense all three levels are referred to as enterprise architecture. In a strict sense, only the enterprise level is included in that term.

An Enterprise Architecture Practitioner's Notes: Volume 3 Solution Level Architecture

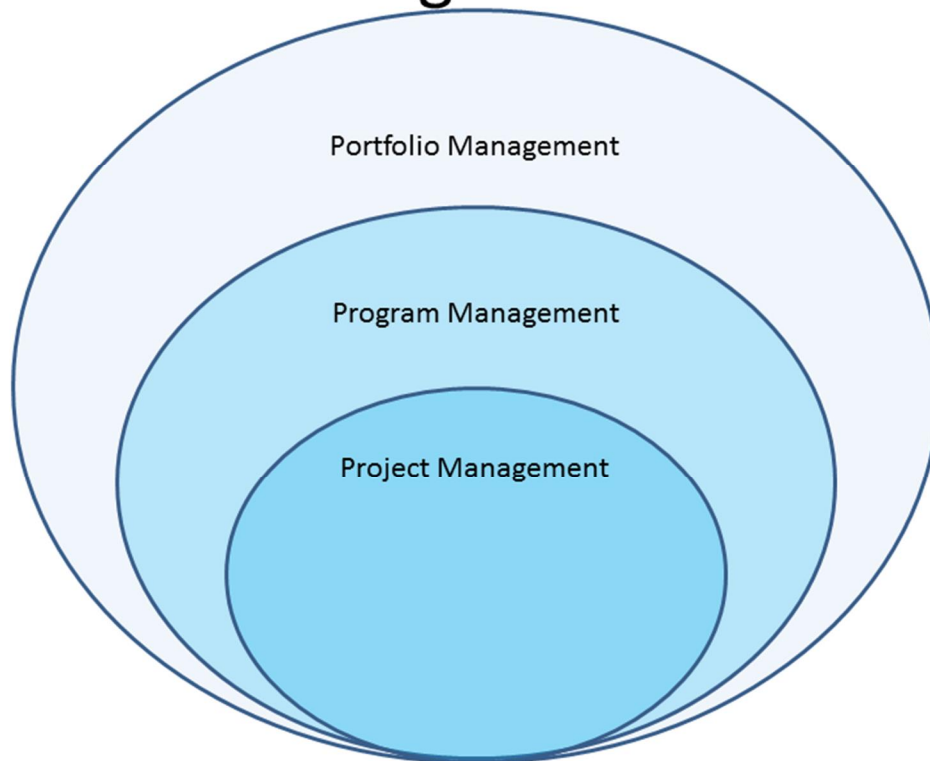
I have written a paper connecting this 15-plus-year-old model with the two other most common activities, EA governance and EA internal practice maturity. You can find that here:

http://www.unauthorizedprogress.com/images/EA_as_5_activities_2014.pdf

I hope this helps.

1.4 CORRESPONDENCE TO MANAGEMENT ACTIVITIES, NEVER POSTED

PMI Management Levels



In the previous section, we saw the three levels of enterprise architecture described by Burk at OMB in 2006 and originally introduced in the early FEAF circa 1999. These have a direct correspondence to the three levels of management described by the Project Management Institute™ (PMI) that are the fundamental subject matter of the PMP™ certification.

PORTFOLIO MANAGEMENT

The portfolio management activity is described by PMI as responsible for business leadership, alignment, value orientation, program selection and portfolio adjustment. This corresponds closely to the enterprise architecture activities described in US OMB Circular A-130 and described by OMB/Burk in 2006.

PROGRAM MANAGEMENT

PMI describes the program management activity as providing business sponsorship, ownership of benefits, benefit streams, comprehensive ownership of the business system, and multiple projects. This corresponds well to descriptions of segment architecture and the improvement activities of a line of business.

PROJECT MANAGEMENT

Project management is described by PMI as providing delivery of capabilities, budget and schedule. Solution architecture as commonly describes support those items. This includes descriptions by Burk at OMB.

MERGING

Architecture Level	Scope	Detail	Impact	Audience	Customer Management Level
Enterprise Architecture	Corporation or Agency	Low	Strategic Outcomes	All Stakeholders	Portfolio Management
Segment Architecture	Line of Business	Medium	Business Outcomes	Business Owners	Program Management
Solution Architecture	Function or Process	High	Operational Outcomes	Users and Developers	Project Management

If we merge the information in the OMB table (previous section) with the PMI information, we get a table like the one above. It implies that the enterprise architecture activity supports the portfolio management activity and the portfolio manager. Further the segment architecture activity supports the program manager. Lastly the solution architecture activity supports the project manager.

While these are not absolute rules embedded in any law or policy, they seem to be important guidelines.

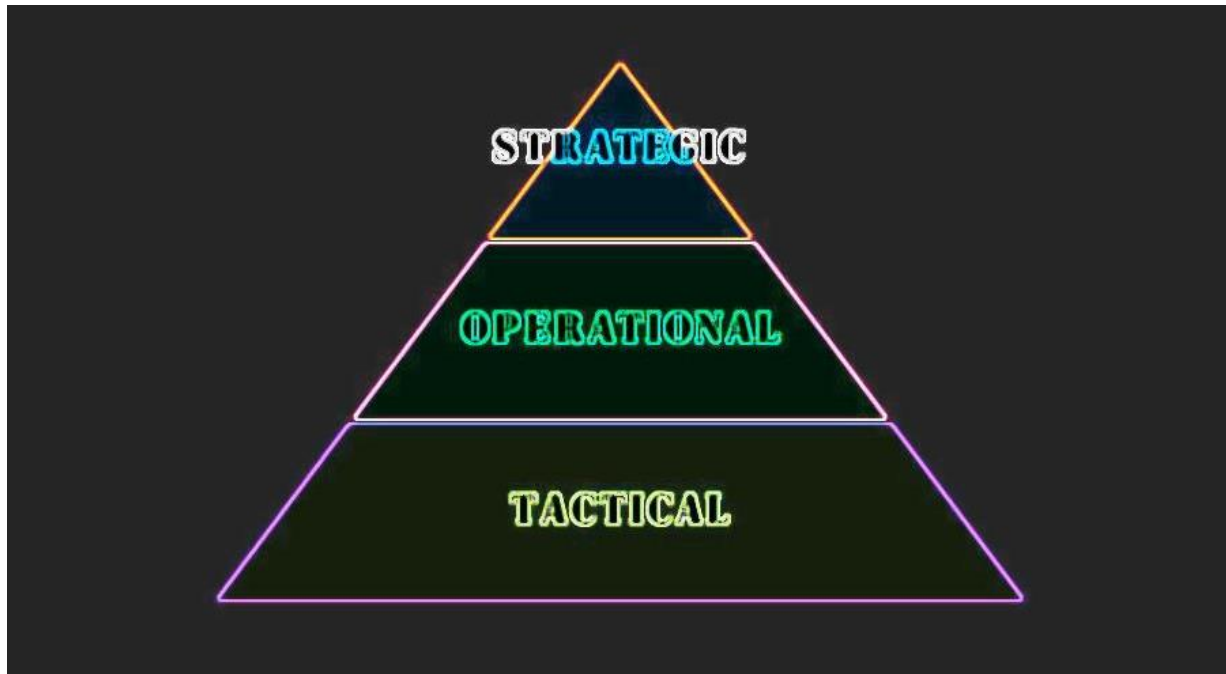
MEASURING SUCCESS

Customer Management Level	Architecture Level	Scope	Detail	Outcome Measures
Portfolio Management	Enterprise Architecture	Corporation or Agency	Low	Strategic Goals & Objectives
Program Management	Segment Architecture	Line of Business	Medium	Organizational Performance Measures & KPIs
Project Management	Solution Architecture	Function or Process	High	MOPEs, MOPs and KPPs (INCOSE)

If we add common material on how success is measured to the appropriate levels, we get the table shown above. The correspondences here are rough, as there is overlap between the system engineering measures of the International Council on Systems Engineering (INCOSE) and the common business nomenclature of performance indicators and the Key Performance Indicator (KPI) versus the INCOSE Measure Of Effectiveness (MOE).

Ignoring the semantic overlap, there appears to be a hierarchy of measures that can be used to measure the success of management and architecture.

1.5 STRATEGIC, OPERATIONAL AND TACTICAL THINKING, JUL 19, 2014



Are you really a strategic thinker? Is your plan really strategy? This is probably not so, based on the predominance of mislabeled plans and concepts: Many use the word “strategic” as a synonym for “important.” While strategy is widely acknowledged to be important, the words are not synonymous. Those who misunderstand the term or who misuse it are unlikely to produce strategy.

STRATEGY

Both time-frame and scope are associated with strategy. If it affects the entire organization and covers a period of years, it may be strategy. Examples of strategy include what markets you will compete in and which you will exit; what are your competitive advantages shared across the company; where you will invest in capacity and where you will divest; and the fundamental purpose (mission, goals) of your company or organization.

TACTICS

Tactics involve point approaches to local problems or situations. Tactics may be reusable for a common problem or situation. Tactics are usually rapid compared to strategy and do not describe activities covering years before fruition. Examples of tactics are Standard Operating Procedures (SOPs) and choice of and purchases or acquisitions of services or products.

OPERATIONAL THINKING

This lies between tactics and strategy, affecting perhaps an entire product line but not the organization or the business processes used repeatedly and changes to them. Examples may include new features or improved performance of a product or a single line of products among many.

It is best to align your chosen tactics to support your operational planning, and align your operational planning to your strategy. In this way you achieve your plans.

MISUSE

Now let's examine some common misuse of the term "strategy." A vendor wants you to have a "mobile strategy." This may well be strategic to the vendor, who sells mobile services or devices, but it is not about your market positioning, your markets, or your major investment areas. It is at best operational to you and perhaps tactical. Everyone is using this stuff; there is no competitive advantage. If you waste money and effort on unneeded toys, competitive disadvantage may result.

Six Sigma or Lean or Agile are said to be strategic and may provide competitive advantage. Adoption and implementation of these may rise to a strategic goal to provide competitive advantage, but once adopted these are operational issues.

STRATEGY FORMAT

In the US Government it has become common to create recurring yearly strategies in the form of a list of broad goals, subdivided into concrete objectives, perhaps associated with some performance measures. Supporting policies are often not included (perhaps due to the complexity of their approval). Sequences of actions are left to operational plans. In commercial use a strategic plan may commonly include all three and are more often confined to a single issue.

CONCLUSION

Perhaps the quickest way to indicate your strategic irrelevance is to improperly indicate that your tactical advantage is strategy. Those trained in strategy can spot the difference. Try to use the terms correctly and you may be better respected by your audience. If you are a CxO or vendor, or anyone between, the misuses of the term "strategy" will likely hurt you more than helping you.

1.6 WHY DO ARCHITECTURE?, JUL 5, 2015



I have left out one topic. I am remiss. This particular topic mystifies me, not the answer itself but all the controversy.

I started out my paid career as a technician. I would repair and maintain complex electronic systems designed by others, much of it past the end of its predicted product life. To do this I needed drawings, lists, matrices and documents which are the artifacts of engineering. Without these the complex and expensive systems I was charged to maintain would remain broken as I could not figure out what might be wrong, I could not troubleshoot.

No manager was both so bold and so dim as to suggest that engineering documentation was not needed for my function.

Later, I was an electronics design engineer and manager of engineering. I was charged with producing prototypes and their associated drawings, lists, matrices and documents. Without these, the production engineering function would have nothing to optimize, and technicians no means to repair. The repair function slowly changed, as all electronics became commodity and broken items were wastefully thrown away.

Electronics design moved to the Pacific Rim and I left it behind. No matter how good I was, the exchange rate (Dollar to Yen etc.) became so unfavorable that they could hire multiple engineers there to replace one of me, and for the same money.

Now managers sometimes spoke of outsourcing all design and the provider keeping the documentation.

I began to work on system integration. I would design systems, do site surveys, and produce drawings, lists, matrices and documents. I was paid to do this. Without such documents the large complex systems we produced could not be implemented, fielded. We had to build them and make them work, then repair and maintain them. I had moved from single boards and units to entire groups of computers and peripherals as building blocks, but the artifacts of design remained the same.

No manager responsible for design and fielding of such complex systems argued that the artifacts of design were not needed. But customers who had outsourced the function were sometimes uninterested in those artifacts.

In the latest leg of my career I have managed the enterprise, often working for the customer. Here there are many large complex systems, which may be interconnected. Those involved in building and interconnecting such systems produce artifacts such as drawings, lists, matrices and documents, to manage the design, construction, fielding and maintenance of the systems and their interconnection. The customer is less interested in the artifacts, so long as the provider is doing the job.

Managers today routinely question the need for design artifacts regarding the enterprise.

So I can tell you these experiential facts and opinions from my career:

Opinion: Architecture and design are mostly synonymous, although some academic minds insist on slicing them apart via tiny nuance.

Fact (by observation): If you are directly responsible for designing, building, fielding and maintaining complex equipment or systems and competent in doing so, you will use drawings, lists, matrices and documents. You will not often question their use if you produce more than a prototype.

Fact (by observation): If you are a customer who has outsourced these functions you will not care as much. So long as the provider has done their job you may not care at all.

Fact (by observation): At some point in the lifecycle of complex equipment or systems you will need to upgrade, modify, maintain or otherwise change them. If they are sufficiently complex, and the original designers have long gone, you will suddenly realize that you need those artifacts (drawings, lists, matrices and documents). You will be unable to meet customer needs without them, and you will be out of business with your present technology. On that day you will become aware that you needed architecture or design documentation.

Fact (by observation): This can all be mitigated if you intentionally plan to throw away all you have and start from scratch every few years. (Zachman refers to this as "scrap and rework".)

Zachman identifies two key factors: Complexity and Change. As complexity increases, scribbling notes on whiteboards and post-it notes will fail as engineering methods. When change occurs you will be unable to use your documentation to manage it. This is true regardless of scale.

Complexity arises from producing systems and equipment to aid in achieving complex goals and objectives, BTW. You produce sophisticated tools to support sophisticated use. Excellence, quality, or cost-effectiveness require some sophistication.

So, as a manager of a complex system or equipment you may have a short-term view and expect to move on before the disaster strikes. You may then argue against engineering documentation, as it is an expense for an item you will not require before transferring the obligation elsewhere. It will not be needed if all your technology and its configuration is commodity to be thrown away repeatedly.

You do architecture or design of an item not for its own sake. You do it so that the item, be it process or box, supports effort to produce product or service. That way you make money, for example. You initially produce it so that it exists at all, then later so it is better. You will stay ahead of competition that way. You want to produce that process or service with lower cost, higher quality, or higher throughput. These are outcomes, to produce the product or service with higher throughput, quality or lower cost. To simply produce, it is a capability.

If you are the customer, buying the product or service, you may discard the documentation as if it were a microwave oven or dishwasher in your home. You will assume the manufacturer has a copy. However if it was custom built for you they may not keep your documentation. If you fire the first vendor, the second vendor will need the documentation. If your enterprise falls apart after the last person in charge has left, the new person may also desire documentation. If the mission or market changes, you will want the documentation. When you cannot serve the mission or the customer any more with the undocumented mess that has been left behind, you will realize you need the documentation.

If you do not believe these drawings, lists, matrices and documents are required for your large and very complex project, system, equipment or enterprise then I request you do one of the following:

Go work for my company's competitors or hostile nations and apply your views there;

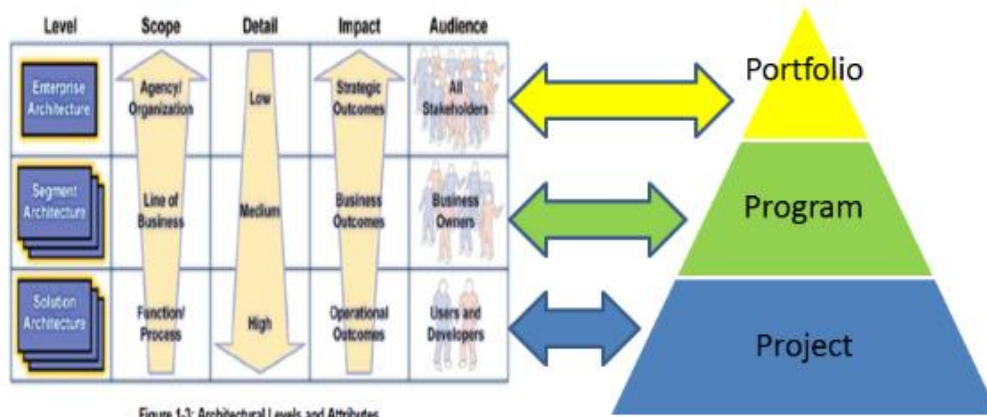
Get out of the way of those doing required work, even if you do not understand it, as your presence is not constructive;

Stick around long enough to take responsibility for your lack of foresight;

Take up used-car sales or tele-evangelism, go far away and quit playing with managing technology before you damage something.

1.7 INTRODUCING ARCHITECTURE TO YOUR ORGANIZATION, JUNE 7, 2015

Architecture and Management



Left part of figure adapted from 2007 FEA Practice guidance, OMB.

Have you heard the story of the Emperor of China asking the great sage Lao Tzu to explain all the secrets of the universe, while the emperor stood on one foot? An impatient and impetuous young emperor thinking it was all that simple is the point of the story. I was recently asked how to implement architecture in a huge contract gone rogue, with many systems thrown together at the last minute to meet deadlines. They seemed to want a simple formula.

Let's assume you have no architecture practice, which is probably untrue and people are probably building architecture in little dark corners because they have to, but let our assumption stand. In my mind, the first step to introducing architecture to your organization is to understand that there is not one architecture. I have listed elsewhere the five activities within enterprise architecture, and so near as I can tell they have different purposes and processes, and implementing each is different.

ENTERPRISE LEVEL ARCHITECTURE

The enterprise level architecture practice supports portfolio management of transformation investments, it has a yearly cycle, following the budget. Its main risk is "boiling the ocean" or trying to do too much architecture centrally. Burk described this level as most strategic, broad but shallow.

Here the term "enterprise" is specifically applied to the whole organization. I am not using the broad meaning, all of this stuff, at all levels, as that would not aid description.

Oddly the startup of this level may be easiest to describe. Rob Thomas II and company described this well for the Federal Government in a good document: [A Practical Guide to Federal Enterprise Architecture](#). Read that.

SEGMENT (LINE OF BUSINESS) LEVEL ARCHITECTURE

Someone must first divide the enterprise into lines of business, and cross-cutting efforts. This must be done at the enterprise level, and everyone must agree to the divisions. Then you can kick off each one of these efforts, associated with an overall transformation program for each.

The segment or LOB architecture serves the program. A clear, 1 to 1 relationship is best. To do this well I suggest you ignore the FSAM and DSAM of the US Federal Government and instead look to the "Mission Architecture" or "operational architecture" efforts of DoD in DODAF. They do this well. An example can be found here:

http://www.rand.org/content/dam/rand/pubs/research_reports/RR200/RR261/RAND_RR261.pdf

Segment or LOB architecture effort should produce business cases proposing needed operational improvements. This level of architecture has moderate breadth, and moderate depth. As for how to proceed, the [FSAM](#), the [DSAM](#) and [DoDAF](#) all have processes for that. There will be one to many segments in an organization.

SYSTEM OR SOLUTION LEVEL

Systems are oddly problematic to define, and you have to choose boundaries. Once you do, you can define them through architecture. This kind of architecture serves a project intended to construct the system and make it operational. The architecture then supports maintenance and operations. There may be many systems in a segment or LOB. Not all may need the same level of architectural documentation.

How do you proceed? First get an [SDLC](#) or equivalent like [PMBOK](#) or the [TOGAF ADM](#) (first used in TAFIM years ago to describe system construction). Then follow that. Stop by [INCOSE](#) in your search.

ARCHITECTURE GOVERNANCE

To institute architecture governance is a bit tricky. It must fit into your corporate governance, or customer governance. The main problem in this is reigning in runaway governance, as everybody wants the power to control something. Go for simplicity. The governance glues the levels above together. Have a look at [my post here](#) for a minimalist viewpoint. Here is a guy with a [5 point approach](#). You can find advice all over the web, but try the early FEA documents to understand how to get started related to architecture.

Starting the architecture above and the governance that uses it is a "chicken and egg" problem. Start somewhere and keep moving. Do not wait for all conditions to be perfect.

MATURITY MANAGEMENT

How do you institute process improvement of architecture itself? You need an independent process. Look [here](#) for some guidance. There will be audits, self-assessments, analysis, and process improvement. (You can do it yearly in the lull between budget cycle efforts at the enterprise level.)

GENERAL RULES

Here are some general rules:

Architecture is lists, matrices, drawings, documents. Do lists first, documents last to the extent you can.

FEAF v1 points out that in architecture, business drives data drives systems drives technology and infrastructure. The original NIST model puts middleware and data exchange between business and systems, and databases between systems and platform. Follow advice like this as to what precedes what.

Zachman often says don't do all the architecture for everything, do it as needed. Do that.

Do architecture at the lowest level that makes sense. The upper levels drive the lower, but are shallower. The upper levels only become shallower at the top by doing architecture at the lowest level reasonably. Also, by performing system architecture at the project or system level, you get architecture that is more relevant to the system and more usable.

STAFFING

At the enterprise level the basic FEAF v1 team has about 6 persons. You have 1 for the inventory of business functions (top level processes), another for the list of data assets, another for the list of systems, another for the standards and approved products, a performance architect and a chief. Tiny organizations may need less. You can hit ten if you do some other sophisticated things, like governance and maturity management and performance management. If you need far more than that you may be "boiling the ocean".

At the segment or LOB level you will have a large team at first, and then less later. This can be 20+ people to describe the architecture in terms of production line, supply chain, distribution chain, value chain, product mix and more. Choose what you need but do not skimp on the initial staffing. The ROI is high here, and this affects your core businesses.

A system can often have one architect. Several systems may share one. Large database centered enterprise software may have 2 or 3 (system, software, database). Unless you have a system of systems, or SOA environment, or some such, more is not advisable; and if you do have SOA or ESB or SOS it is probably a segment and you screwed up the system boundaries. (All the ESB, EAI, ETL, data-warehouse and DataMart efforts are best mixed into a single program for several reasons.)

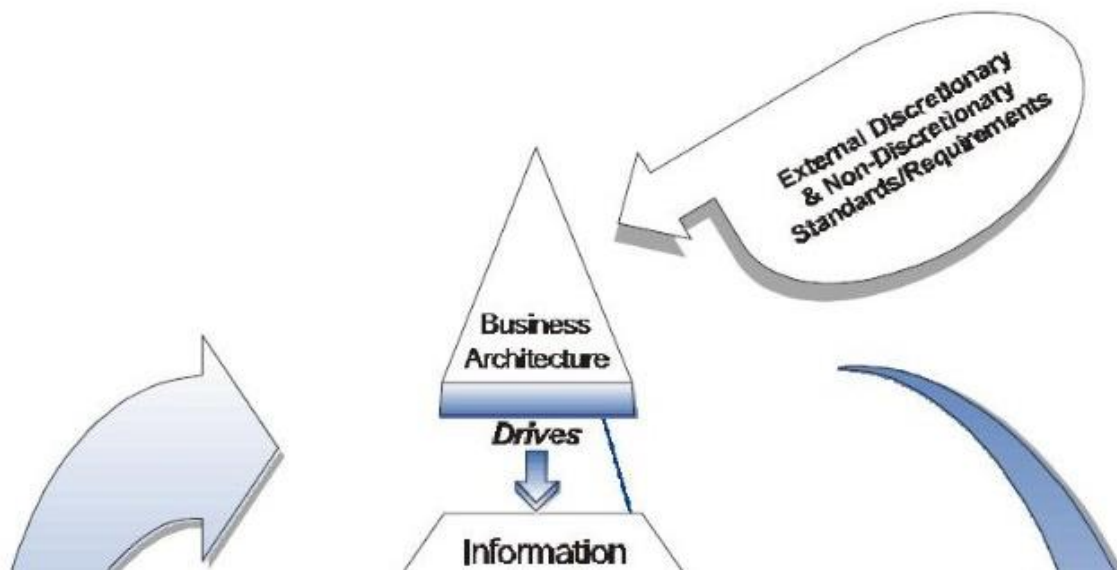
Somewhere, above all this effort, you need a chief architect or chief systems engineer to sort it out-maybe both. Don't skimp on this guy.

If you have all these folks and more, and nothing is getting done, the organization and scopes and authorities are not clear. It's a management problem, not an architecture problem. Fix it.

CONCLUSION

Initiating an architecture practice in your organization is not a single, simple formula.

1.8 EA, BA, DA, CARTS AND HORSES, JUNE 28, 2015



Partial figure from NIST SP 500-167, US Government

Recently I saw a presentation from a BPM guru that completely mischaracterized EA as IT only architecture. The ignorant presenter was also not aware that Business Architecture is a component of Enterprise Architecture. This rendered the presentation worthless.

I have seen Business Architecture create a body of knowledge emphasizing such misinformation, and organizations elevating business architecture outside of Enterprise Architecture. Such a huge step backwards will not create a seamless application of technology to the mission, or in support of strategy.

Separating BA from EA is not constructive to the management of either technology or modern business.

Data Architecture has also been oft promoted to outside of Enterprise Architecture. This also rewrites history, and works against a unified architecture.

The value of architecture will not be achieved by attempts to split it into fiefdoms. Holism is a key concept. Examples of splitting out this and that component and giving it dominant political power or supremacy over EA holism are also examples of bad management.

If you want real results, keep the components of Enterprise Architecture in their context and relative position within it. The relationship was described in the first EA document (NIST SP 500-167), and in the early FEAF, and is implicit in DODAF, etc. Do not mistake political power grabs with effective management or effective architecture.

1.9 ACHIEVING SUCCESS, APRIL 12, 2015



Success is one of those words that loses meaning without a context. People argue endlessly regarding what success means, and it is meaningless without the key. This post is for younger folks and anyone else who do not have that key.

I don't want to string you along: There is no success without a goal. When you achieve your goal you have success. There, the mystery is gone. You can look at me, and plenty of old guys, and say "he is no billionaire, he is not successful". Hey, that was not my goal. If that is your goal, the same approach will work. Go for it.

To make your life successful, follow the same advice that generations in the past have followed. Here it is: use the planning cycle. It's not obsolete. It's not irrelevant, because our time is full of chaos and turmoil. It's not optional. It's not a peripheral part of the life of the driven and successful. It is a core activity if you want to succeed.

Let me describe the cycle you use for success:

- Set (or adjust) a goal or goals. Pick carefully. You are going to achieve these if you pick obtainable goals, and you need to appreciate them once you have attained them. These goals must reflect who you are, your values, what has meaning to you - as they will be all that for you in the end.
- Make plans. Include contingencies, risk mitigations. The harder the goal the better the plan must be. Make plans that are real, concrete, and attainable. Do not be timid. Be objective.
- Execute the plan. Be brave. Stick to it. Never give up. Work hard. Ignore distractions.

- Evaluate and consolidate. Check where you are versus your goals. Evaluate the effectiveness of your plan. Be brutally honest. Be factual. Then go back to the top.

Pick something like a yearly schedule for the cycle. There are other descriptions of the cycle, most are fine. Now go do it.

Here is the hard part of this advice: Others who have done this tell me that there are very few things in life more important to your success. They say learning to read well, study, write well and speak well may be such a thing, but it can be achieved by this cycle. They tell me you can be tall or short, red or green, LGBT or straight, woman or man, Catholic or agnostic, born poor or born rich, big or small, ugly or good looking, and this approach will work anyway. So do not whine and complain about whatever disadvantage life has given you, we all have disadvantages and things to overcome. Shut up and get moving. Stop waiting for someone else to fix your life, it's yours to fix by achieving goals. If someone does help you, be grateful but do not expect it or rely on it.

Some may have achieved more this way, some a bit less perhaps, but still they achieved success. This advice works for almost anyone, as well as for companies and government organizations. It applies to whole countries and civilizations. If you have no goals you are unlikely to achieve them. If you do, you have a good chance.

SECTION 2: SOLUTION LEVEL ARCHITECTURE:

Section 2: Solution Level Architecture:	30
2.2 Integration, System Engineering & Enterprise Architecture, January 11, 2015	33
2.3 What is System Integration?, August 23, 2014	35
2.4 Enterprise Architecture vs Systems Engineering, August 16, 2014	38
2.5 Three Views of System Engineering, January 10, 2015	41
2.6 Enterprise System Requirements, December 13, 2014	42
2.7 Allocating Requirements, July 12, 2014	46
2.8 Basic System Test & Evaluation, March 24, 2015	48
2.9 Choosing Components, Sep 1, 2015	50
2.10 Drawing the Box, June 22, 2014	51
2.11 Drawing Boxes, December 19, 2014	53
2.12 Managing Interfaces, February 13, 2015	55
2.13 Functional, Virtual and Physical Architecture, Sep 1, 2015	57
2.14 System Architecture Quality, February 21, 2015	59
2.15 Website Solution Methodology, March 22, 2015	61
2.16 About Enterprise Software, December 24, 2014	63
2.17 The Value of Enterprise Software, August 3, 2014	66
2.18 Centralization vs Decentralization, Aug 19, 2015	68
2.19 The Very Largest Distributed Systems, September 14, 2014	73

Posts 2.2 to 2.5 try to establish terms and concepts. Posts 2.6 to 2.8 describe requirements and their testing. Posts 2.9 to 2.13, 2.15, 2.18, and 2.19 discuss how to do architecture. Posts 2.14, and 2.16 to 2.19 speak of architectural “goodness” and applicability.

Questions for Section 2:

1. Why is system (solution level) architecture a subset of systems engineering?
2. What are the characteristics of a good design?
3. Why does functional architecture come before more concrete efforts?
4. Does your local college have a course in systems engineering? Does it cover architecture?
5. What is the difference between software architecture and systems architecture? Are they interchangeable?
6. Did you ever design anything very complex? Was it hard? Did the parts fit together on the first try? Did it operate on the first try? Did you plan? Did planning help?
7. Is testing important? Is evaluation important? Why?

2.1 ARCHITECTURE: SOLUTION VS ENTERPRISE, AUGUST 30, 2015



As I have written elsewhere there is both a broad meaning and a narrow meaning to the term "enterprise architecture". Some use it as a catch-all phrase for all the related architecture activities an organization engages in (especially any involving big IT or "enterprise software"). Some use it as a specific term for those activities supporting the CxO level and the portfolio of transformative efforts.

For a moment, let's examine the specific meaning of "enterprise architecture", which I sometimes distinguish as "enterprise level architecture", from "solution level architecture" (or simply "solution architecture") aka "System Architecture". Here it is, the difference between "enterprise architecture" and "solution architecture".

DIFFERENCE

- Solution architecture is about a single system, perhaps produced by a single vendor. Enterprise architecture is about all the systems in the enterprise, not one, the portfolio of all the systems, from all vendors. This is one versus many.
- Solution architecture is about reusable software, patterns, objects, platforms, programming and product choice, cloud development tools, distribution and release, integration, and internal interfaces. Enterprise architecture is about redundant applications, redundant databases, alignment to the mission or strategy, approved products, standards for all solutions, measures of performance and goodness.

- Customer organizations do enterprise architecture and some solution architecture for those solutions built internally. Vendors of solutions and software rarely, almost never, do enterprise level architecture. Instead vendors do product management. The enterprise the vendor is most interested in is that of the customer, not usually their own. If the vendor did enterprise architecture they would look at the set of systems they have purchased and now use, not those they sell.

COMMONALITY

- Both have component architecture addressing business use, databases and structures, applications (and software), infrastructure (networks, platforms and interfaces), standards, and security.
- Both deal with systems. Your product(s), as configured, integrated and bid, is a system. The whole enterprise is a vast and bigger system in which the solution you have bid (vendor) is a small component.
- Both get lumped together under the broad blanket definition of "enterprise architecture".

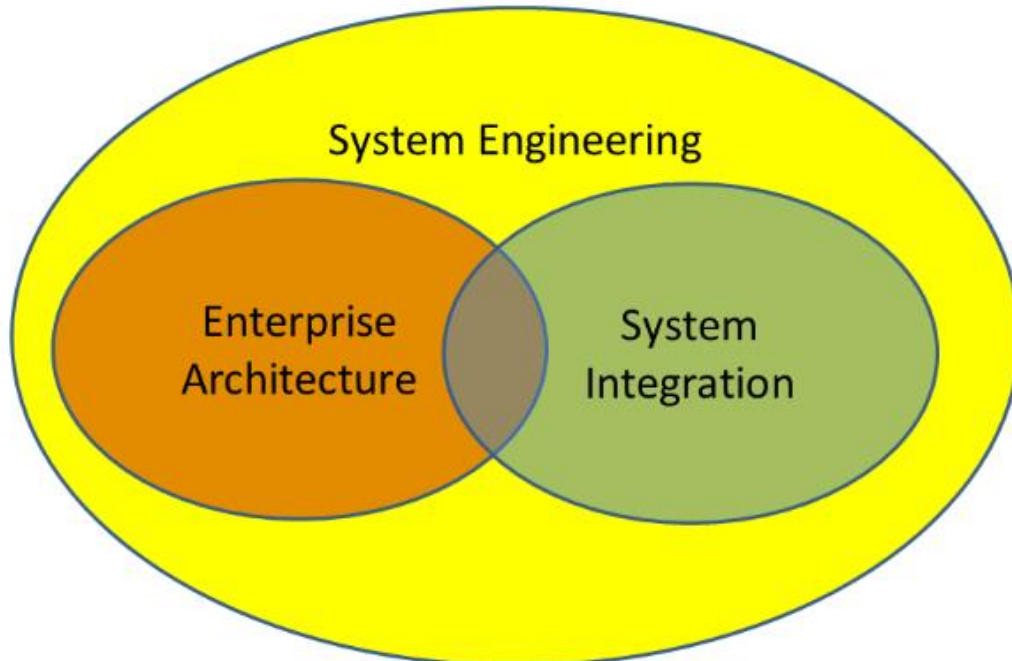
CONCLUSION

While there are common terms, methods, principles the two practices are very different. Referring to "solution architecture" as "enterprise architecture", without being aware of the more narrow use of the term, can lead to miscommunication, confusion and conflict.

I didn't create the definitions. I do not control them. I simply describe the situation as it is.

(The picture BTW is me touring the remains of Slain's Castle, jeans tucked up to avoid the dirt and sand and poison ivy. I suppose the design of one castle might be seen as solution architecture, and the relative positioning of all the castles in Scotland as enterprise architecture, supporting strategy. I was struck by how small the doors and hallways were in some of these castles, although my girth has grown in the last few years it is my shoulders that did not fit. Also I was too tall for several. They were smaller then, those Scottish Knights. Solutions change, requirements change. Now we don't need castles.)

2.2 INTEGRATION, SYSTEM ENGINEERING & ENTERPRISE ARCHITECTURE, JANUARY 11, 2015



What is the relationship between Integration, System Engineering and Enterprise Architecture? The relationships are hard to sort out, but here are some facts:

The first paper to contain the phrase "enterprise architecture" was part of an effort to contain integration costs.

Architecture (presumably DODAF) is an explicit step within the scope of the system engineering V in DoD.

Enterprise architecture is thought by many practitioners to oversee integration technologies like SOA and ESB.

Martin created a concept called "enterprise engineering" which is a superset of enterprise architecture, and explicitly identifies system engineering as a predecessor.

Enterprise architecture is mandatory across the US Federal Government via the Clinger Cohen act and OMB Policy A-130, but system engineering is not.

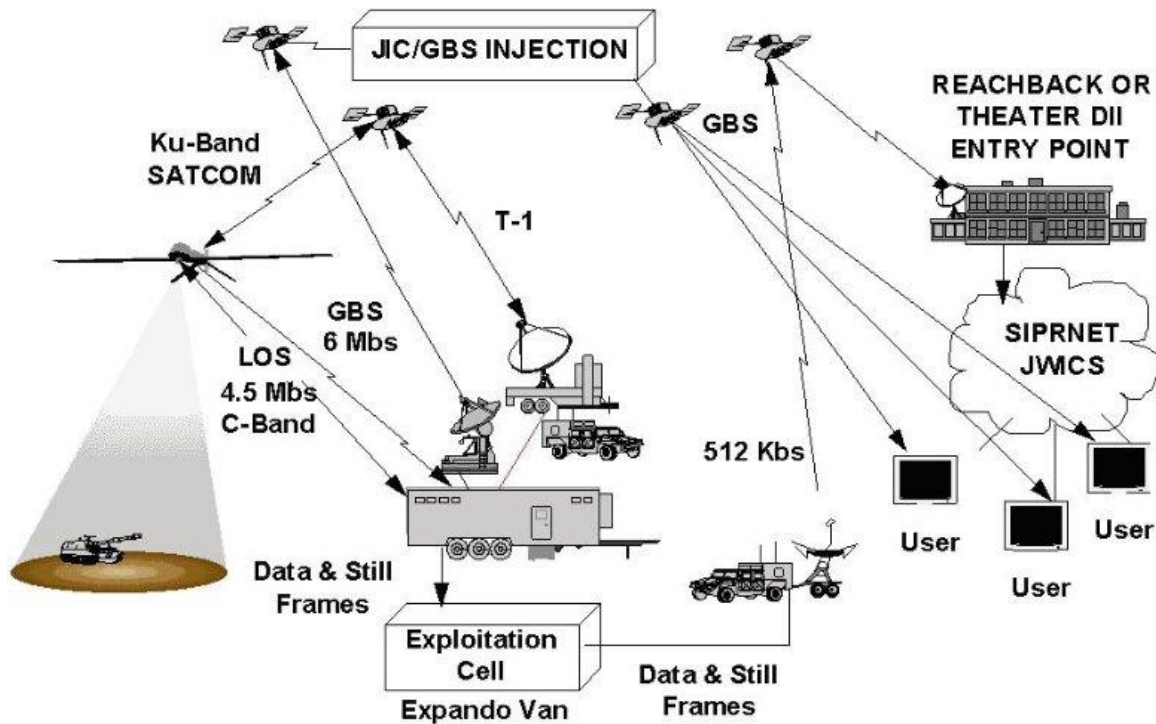
FEAF and FEA contain no references to system engineering.

My own opinion is that enterprise architecture is a proper sub-discipline of system engineering. System engineering, elevated to the enterprise scale and scope, is the context of enterprise architecture. However some papers and concepts may be absent in making that reality in academia and in practice.

An Enterprise Architecture Practitioner's Notes: Volume 3 Solution Level Architecture

I also believe integration is a subset of system engineering. For whatever that is worth, my posts reflect that thinking.

2.3 WHAT IS SYSTEM INTEGRATION?, AUGUST 23, 2014



Picture from DoD program, US Government Image.

What is System Integration? People often speak of it, but many do not really know what it is.

Example

Imagine the set of applications used at a large customer site. This might include a whole ERP suite, CRM, and a dozen custom applications that perform the core business functions. Six of those custom applications may be very old, and the source code is lost. Integration is the process of making those applications work together and share information.

Custom Application Development

Custom software development is not directly a part of integration. It only becomes a part of integration when some portion of the needed functions are not present. If between COTS applications A, B and C there is functionality left out (D), then integration includes developing D as a side effect. Sometimes the construction of a single application that has many interfaces to complex external hardware will be identified as system integration; this is beyond simple application development.

Continuous Integration

Note that the term "continuous integration" has little to do with integration. It is the practice of merging new application software into a common code-base several times per day. That is part of custom application development, and has very little to do with the overall scope of IT or system integration. Don't be fooled.

Integration Technologies

There is a long history of tools to aid system and IT integration. In the beginning we used tapes and drum, and later shared disk to integrate batch programs that were interconnected by JCL (Job Control Language). Later we used shared databases as the medium of exchange between multiple applications. We created UNIX streams and MS pipes and RPCs and IBM APPC and more to move data between applications. We developed common environments such as CORBA and .NET after that to aid interconnection of programs. We developed workflow systems. We developed dozens of brands of middle-ware. Today we have JMS (a bit of a throwback), EAI, ESB and SOA to integrate between applications.

Back-end Development

Some back-end development is common in integration. It is especially common to code adapters to interconnect things. The idea of the workflow engines embedded in EAI, ESB is to reduce or eliminate other programming in back-end integration.

Interface Discipline

Integration is all about the interfaces between disparate bits of software, often from different vendors, mostly legacy. How do you create, control, manage and maintain those interfaces? How do you sequence activity between the software across the interfaces? How do you negotiate interfaces to software in other organizations? How do you create governance to manage the upgrades and reuse of those interfaces?

Workflow Software

In the 1990s workflow software was everywhere, now it is mostly a component of your ESB or EAI software. The intent of workflow software was to provide a visual scripting environment that business analysts could use without need for a programmer (wherein the adapters had already been coded in a reusable way). WARIA the workflow industry group studied, found and marketed that whereas 80 percent of software development efforts failed, that 80% of workflow efforts succeeded. This allowed drastically reduced programming systems integration.

Unfortunately BPMN standardization efforts that never worked, Silicon Valley companies who never understood workflow, and vendors that undermined the elimination of programmers, have reduced many current workflows systems to an excuse for even more programmers - not fewer. You can still get some workflow systems that eliminate programmers in the "glue" between systems.

Enterprise Architecture

The discipline of Enterprise Architecture was developed mainly to manage the complexities and costs of IT integration at customer (consuming) organizations. Later we found that to do this, each

bit of software had to serve the mission or strategy; otherwise you had many costly stray bits of excess junk in your enterprise. The portfolio of systems within the enterprise had to be managed to eliminate redundancies and ineffective extras. With peripherals like bar-code scanners and manufacturing robots, enterprise architecture and integration can stretch the limits of IT. Add drones, and EA faces an expansive future.

SOA and EA

Service oriented architecture is integration between applications, and inherently part of EA. SOA governance is a part of larger EA governance. Doing these separately is wasteful and ineffective. Success can only be achieved, in either case, by coordination or merger.

SoS

The term System of Systems has been coined to describe this situation where you have many applications from many vendors, mostly legacy, that may not have been designed to inter operate with each other, but you interconnect them anyway. Usually creating an SoS requires or is aided by an integration technology.

Recap

Integration is the interconnection of discreet applications or systems, often by different vendors, most often legacy software, usually designed to work independently, and the management of that activity. Integration is about interfaces and integration technologies. One term for a set of interconnected independent applications is a System of Systems. Enterprise architecture was developed to manage integration, and grew to encompass support of the IT portfolio, strategy, and ultimately the portfolio of transformation efforts.

2.4 ENTERPRISE ARCHITECTURE VS SYSTEMS ENGINEERING, AUGUST 16, 2014

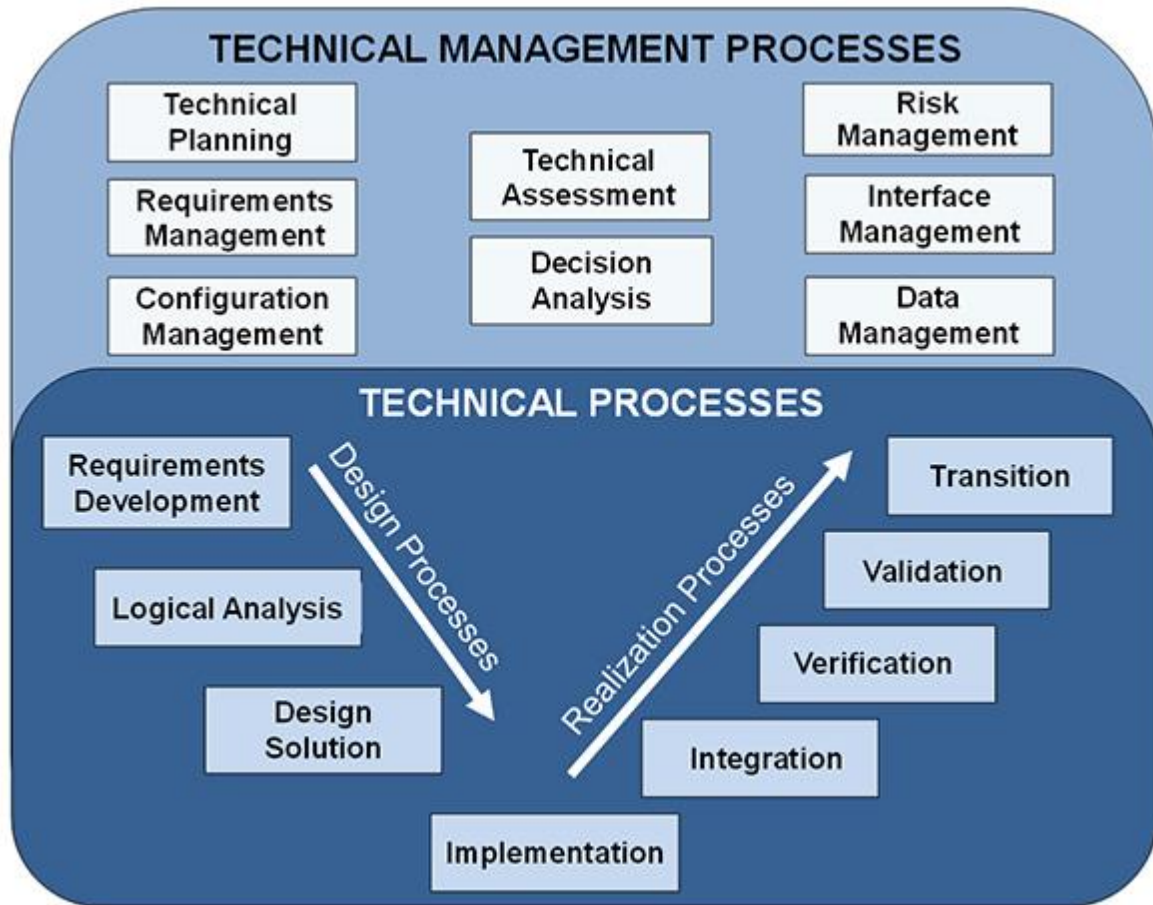


Image from Department of Defense Training Material.

What is the relationship between system engineering and enterprise architecture? Where does one stop and the other begin? This is a topic of some considerable discussion by experienced analysts. Here is a quick introduction to some key points.

Solution Level Architecture

The diagram shows the DoD system engineering V, as depicted in DOD system engineering training. You can see on the left the box labeled "Architecture Design". As the image depicts the full scope of system engineering one may infer that "Architecture and Design" is a proper subset of system engineering. From such DoD training, we know that the architecture discussed often is produced using DODAF. This approach has been used much and works pretty well, so let us accept it as a valid bit of thought to rest our introduction on.

For the solution level of architecture, solution architecture can be thought of as a subset of system engineering.

In practice certain other system engineering processes I have written of, including "drawing the box" around subsystems and allocation of requirements, are commonly part of architecture. The production of stakeholder requirements and derived requirements from analysis is specifically outside the architecture box as depicted.

Segment Level Architecture

Burk 2006 in the FEA Practice Guidance from OMB describes 3 levels of architecture. The middle level is segment architecture. To extend our DOD line of reasoning, the BEA (Business Enterprise Architecture) of DoD is arguably analogous to segment architecture. A "Mission Architecture" in DoD is again arguably analogous to a "Line of Business" architecture in the FEA as well.

Such architectures describe the working parts and their relationships. In DoD the DODAF is most often used to describe these architectures. However I have not found much discussion of system engineering in this context. On the other hand, a mission or "Line of Business" is definitely a system and amenable to analysis and engineering.

I will not explore this here, but you can link this discussion (written here) to the notion of a "System of Systems".

We do know that the scope of the mission, its goals and objectives, its constraints and compliance needs regarding policy, are delivered to architecture and not produced by architecture. This is analogous to requirements development being outside architecture at the solution level.

Let us say then that, in theory, architecture is again a subset of system engineering at the segment level of architecture. However, in practice we are perhaps short of some description and guidance in how that works. But remember, the system engineering diagram depicted above is not intended to be restricted only to the solution level in DoD!

Enterprise Level Architecture

Here we leave the DODAF behind. In Burk 2006 the enterprise level is associated with the portfolio across the enterprise. In DoD the set of all standards is in a repository that is not formally part of DODAF. Further the set of all systems and their architectures is in another repository not formally part of DODAF. The set of all databases in DoD is not formally tracked. The set of all transformational programs in the portfolio is not formally tied to DODAF in any strong way.

The FEAF, the FEA, and Zachman better relate to the portfolio of all transformational investments in the enterprise than does DODAF.

Our guidance from DoD will help us less here. This is not to say DoD does not have management mechanisms at this level, they do have these- but they are not formally part of DODAF.

So where can we look for guidance on how system engineering and enterprise architecture relate? Martin wrote a book on something called "Enterprise Engineering". It explicitly identifies drawing

from system engineering. It has a scope that is wider than most enterprise architecture frameworks claim, as it claims organizational culture and continuous methods as within its scope. So we might say that enterprise architecture is a subset of Martin's construct that draws on system engineering. This is perhaps a weak argument.

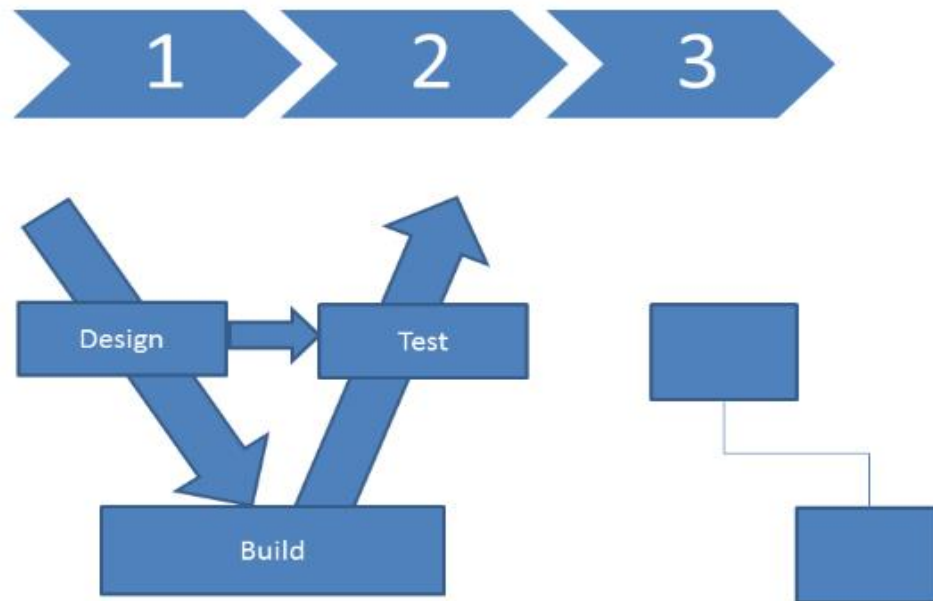
We can also note that the strategy of the organization, its mission or goals and objectives, its policies, are delivered to enterprise architecture and not produced by enterprise architecture. This is how it happens in practice, in my experience anyway. You could claim a partial exception for the "IRM Strategic Plan" aka "IT Strategic Plan" required of the CIO in US Federal Government, but no further. This is analogous to the requirements being handed to architecture at the solution and segment levels. (At least one book has this backwards, though.)

You can link this, again, to the notion of a "System of Systems", and SOA. I will explore that elsewhere.

Conclusion

We can probably say that in theory enterprise architecture is a subset of system engineering. We can probably say that a concrete standardized conceptual framework for this across all three of Burk's 3 levels of EA is missing or not well enough known to have come to my attention at least. (I hear some are working on this problem at Steven's Institute of Technology.)

2.5 THREE VIEWS OF SYSTEM ENGINEERING, JANUARY 10, 2015



When faced with teaching someone about system engineering, most break out in a rousing chorus of the merits of "system thinking". Yea verily, but it is a bit overdone. I focus on imparting three primary practical views of the subject that you will need to accomplish the work.

SDLC

A System Development Lifecycle shows which activities come before others in building a complex system. This can save money in development.

I dislike the teaching approach that says the SDLC dictates what steps must be performed, and prefer that where, if you choose to do something, do it in this order.

V model

A V model shows design activities and corresponding test activities. The best V models show one test per design activity, and one document shared between them. For example functional design produces a functional requirements document used by functional test.

Theory

Ludwig Von Bertalanffy gave us the notion of the open system: A closed environment but with well-defined interfaces. He showed that such environments with interfaces can be the basis of analysis - just as closed environments without interfaces had been before.

Beyond these basic 3 topics the student can begin to understand system thinking.

2.6 ENTERPRISE SYSTEM REQUIREMENTS, DECEMBER 13, 2014



How do you write requirements for an enterprise software system? Well, the process is different than for a product (I have done both). I recall drawing out a business process and then writing all the requirements for a moderately sized solution, myself, in a day. Then writing out a proposal for it is another day. People make this stuff so hard. Here are some tips on the old way of doing this.

Old Days

First, FEAF 1 and 1.1 describe how to design enterprise software in a cryptic fashion. You would have to grab an old-timer to pull it out, but it's not hard to perform once you know what to do.

Start by sketching out the current business process or processes you will automate. Model until all the understanding you require has been found, then stop. After talking to the owner of the process, now map out what the new process should look like. If you do this in BPMN you could spend days trying to get all sorts of extra detail correct, so try not to do that. Try using IDEF0, you might learn something startling.

Now, for each process step or activity, identify the needed data. Be sure to capture any paper forms they use, as they are a goldmine of data used. Use that to sketch out a conceptual data model, mostly just tables and no attributes.

You now have data tables and process. There is a direct relationship between screens (pages) and tables. (Think sitemap.) Those map to steps and roles in the process. Sometimes different roles will need different views of the same data, so watch for that. Now you have a list of screens (pages). Don't leave out the screens (pages) for administrative and security functions.

Requirements

Go back and capture the requirements now. A good analyst can read these right off the process diagrams. (We will use operational, functional and system requirements for our example.)

The operational requirements describe what characteristics the new, changed process(es) must possess.

The functional requirements describe how the new tool will support the new process.

The system requirements will describe how the screens (pages) work.

All three constitute a hierarchy. You can write them as an outline in less than an hour for a system of a few dozen screens (pages).

Draw out the screens (pages), a process called wire-frames today. Put notes on the drawings referring back to the requirements. Now clean all that up, put it in a single document, and call it a technical proposal. Having screens (pages) and requirements, producing estimates per screen is a cake-walk. Usually you could say one page per programmer per day, maybe 2. A screen (page) for a compound table view would be a bit longer. You could easily specify all the message boxes (modal dialog boxes- whatever) right in the initial design.

Comparison

How have things become better? That is an interesting question.

In the 1990s you had an OCX or Java Bean that would allow you to translate database table to code very quickly. Now you are asked to develop layers of code wrapping data objects and implementing business rules that will NOT really be reused, though we will all lie to each other about that.

People try to use use-cases to simplify design. They do not adequately describe process, but people try to replace a process diagram with them. If you replace functional requirements with a use-case it screws up the natural flow of the requirements hierarchy, increasing analysis time, and disrupting any efforts at producing an RTM quickly for testing.

One guy could lay out the complete application in a day or two then, for a moderately sized application. Now we have to sit in dull meetings and debate priorities and descriptions in committee.

At best it's a wash. After watching many such efforts, I believe development costs have actually increased for enterprise software. Reliability and quality seem to have also suffered. The risk of complete failure has been reduced, but the risk of incomplete or misaligned solutions has become near certainty. What works for vendor products may not be best for enterprise applications.

Kool-Aid

So let's try that the Agile way. We need **USER-STORIES**, **EPICS** and **THEMES**.

A **USER-STORY** describes something the user wants in some software, like a feature. Purists may want to write it on a 3x5 card. A report might be an example.

An **EPIC** is a big story, sometimes containing multiple smaller User-Stories. The set of monthly reports might be an example.

A **THEME** is a collection of Epics and Stories. The whole system in which the reports fit might be an example.

OK, we have gotten all the way to **THEMES**, and it sounds like a single project in scope, barely a program if you force it. Whereas operational requirements may drive a large program, Agile Themes are a bit more tactical in scale, IMO.

You can specify software with **THEMES**, **EPICS** and **USER-STORIES**. No one says you cannot. It works fine. However **THEMES** are a collection and not a summary.² It is hard to perform operational testing without some summary of operational impact. Some measures, normally not Agile, are also nice. Here is the rub, if you do Agile things, you have to figure out measurements and testing, and Agile focuses you on the lower (tactical) levels of test and evaluation.

Now we have DevOps which seeks to automate and cast in stone a tactical view of requirements and testing. The lower the level of test and evaluation, the less valuable it is in proving ROI.

Contrast

In the USAF the Strategic Air Command demonstrated evaluation. There were operational evaluations all the time, and an evaluation would not test some technology but instead the organization as constituted of people and technology employed. The Inspector General (IG) or Evaluation Team would arrive at a base and demand to see how many operational aircraft you could get in the air in so-many minutes. If you went to sell SAC some mission-relevant software they might very well ask how many more operational aircraft they would get in the air. (An "I don't know", or "none", gets you a "no-sale" on the old cash-register.)

Is putting aircraft in the air a **THEME**? Yeah maybe, but you have to analyze down a few levels to get to any software. It is not an obvious matchup.

Enterprise

The enterprise puts operations and operational processes first. Some improvement may require some new technology (what Government calls a "material solution") or not ("non-material solution"). You do not buy software for the sake of cool new software. Nor hardware either.

Maybe all you need to get more of those bombers in the air is more gas trucks, or another runway, or a few more people on standby.

Your requirements process should reflect that. Your analysis should allow for the situation where you need not write or buy any new software or hardware at all, perhaps just improving manual processes.

² I think Agile encourages speed over rigor and completeness and hence is great for prototyping, but a minefield for quality assurance, robustness and extensibility. Paul Vereycken.

Conclusion

The state of the art has changed, but it is not that much better.

2.7 ALLOCATING REQUIREMENTS, JULY 12, 2014

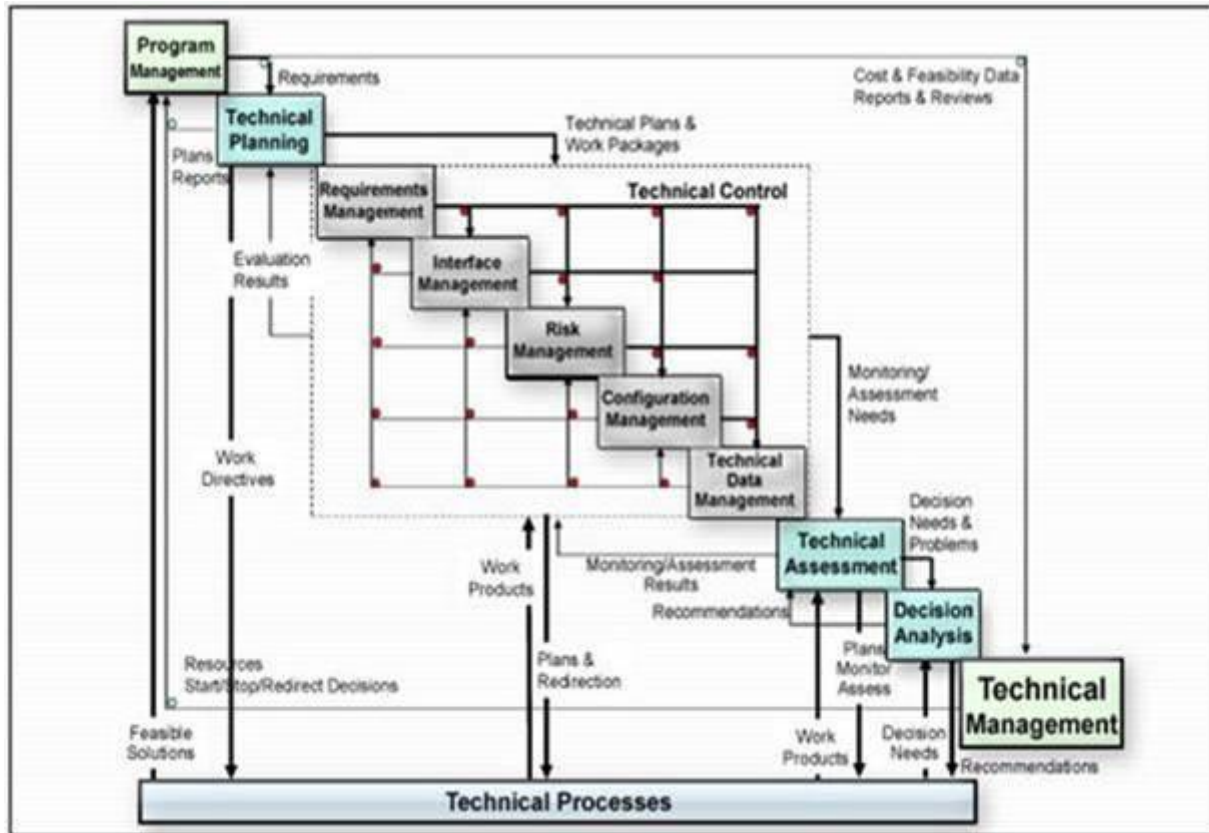


Image from US Government training documents, DoD.

There is a topic in systems engineering that has been ignored lately. Organizations are forgetting how to do it. Kids do not know what it is. This is the process of allocating requirements.

Once you "draw a box" around your system you are then faced with defining subsystems. Choosing subsystems often revolves around internal cohesion and reduction of external interfaces. Sometimes it involves localizing interfaces that should be protected or should be proximate- such as a high speed bus. People still seem to remember how to do this.

The forgotten part revolves around taking the list of requirements for the overall system and deciding which apply to which subsystem. The process is not hard, mostly you inspect each requirement for relevance. Occasionally some additional work or method is required, and often it is problem specific.

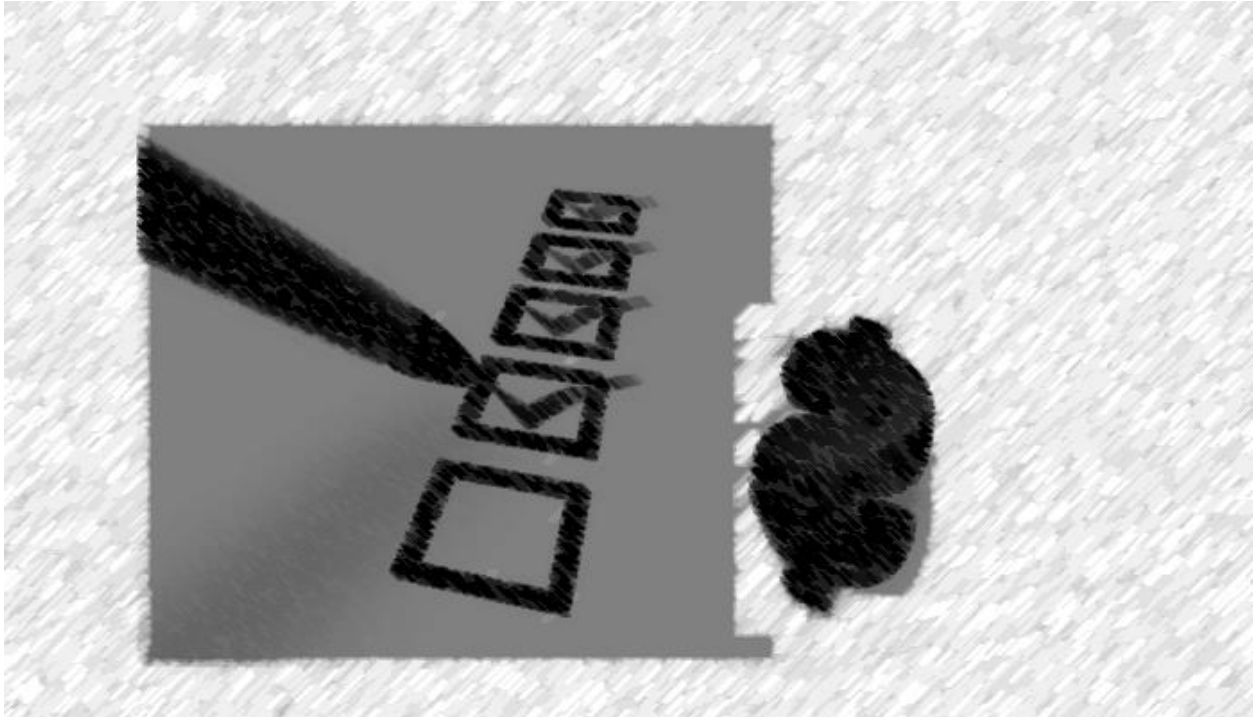
Allocation is not exclusive, a system requirement may apply to two or three subsystems.

Once you have the matrix of requirements by subsystem you can produce a report for each subsystem, identifying its requirements. You can then analyze requirements within the subsystem to the next lower level. Often the team working on the subsystem performs this further analysis. Sometimes the lower level requirements are exposed to and tracked by a function at the higher level of technical management.

An Enterprise Architecture Practitioner's Notes: Volume 3 Solution Level Architecture

This is a basic system engineering topic. All done...

2.8 BASIC SYSTEM TEST & EVALUATION, MARCH 24, 2015



System T&E (Test and Evaluation) for large or complex systems is a topic widely misunderstood and improperly executed. I will present an overview of T&E suitable for managers to improve their T&E operations. The basic concepts and organization are widely accepted and documented. Tests can be very expensive and critical to program success.

The Test Engineering Master Plan³

In any large or complex system effort there will be multiple tests and evaluations of different types. Common practice is to construct an overall plan stating which types of test and evaluation events will occur, when they will occur, and who will perform them. Such a document is not the plan for any test, but a list and schedule of all tests. This is a thin, simple document and excessive detail should be avoided: most detail belongs in the test plan.

The Test Plan

Each test or evaluation has a different, separate, distinct plan. Where a test or evaluation may be performed repeatedly, such a plan may be reused. The test plan should contain all processes, procedures, resource lists, test cases, requirements or use cases, expected outcomes, detailed schedules and similar detail. References to other tests are outside the scope- that goes in the Test Engineering Master Plan.

³ This is synonymous with the "Test Strategy", although some differences in content will occur in various cases. Observation via Dale Chalfant.

The Test Manager

Each test should have an assigned test manager. The manager of the test should be responsible for the overall success of the test, and its execution.

The Test Review

Prior to executing a test or evaluation event the availability of all resources and the plan should undergo formal review. In this review the test manager should receive the final approval to execute the test, or a new review should be scheduled.

Test vs Evaluation

Tests determine if the item under test meets specification. Evaluations identify if the specifications were correct and the item improves operations⁴. The item under test/evaluation changes from the item constructed to the organization when using it, in some sense.

Test Levels

Multiple tests may occur to check conformance to multiple specification sets. Specifications are conventionally nested, with subsystem specifications inside system specifications. Functional specifications are also often nested within operational specifications. Testing should proceed from the narrow, smaller scope to the larger, more general scope. Tests of greater scope usually are recurring, and measure over longer timeframes (often years).

Conflict of Interest

Certain tests are conducted by the vendor prior to delivery. Outside of that set of activities it is important to limit or control the degree of vendor influence and involvement. Watch out for conflict of interest.

Test Replacement

If you adopt certain test methodologies such as those associated with Agile or DevOps, these may replace a certain test or tests. Do not be confused that these methods replace all test and evaluation. The range and scope of such methods tends to be narrow, and focused on lower level tests. Proving that software contains no logic errors, for example, does not demonstrate its operational effectiveness. Do not unknowingly sacrifice traceability for expedience.

Conclusion

If you follow these long proven rules and apply normal project management techniques, your testing efforts will be likely to succeed.

⁴ See post on performance measures in the enterprise, elsewhere in these volumens.

2.9 CHOOSING COMPONENTS, SEP 1, 2015



When choosing components in a system to be integrated or constructed, you do not simply choose the best available for each in isolation. Choosing such components is more like "moneyball" than it is like choosing your ultimate fantasy football team.

The system to be built may have a price-point. It may have functional limits to fit a market niche. And you must think of form factor. Other aspects of the product positioning are chosen through a process of "trade Space", which manages tradeoffs in systems engineering.

Most importantly you need to look at the adjacent components, those connected to this by some interface within the system, and assure that the interfaces are compatible or require a minimum of adapters, special tools and duct tape.

To do this you need a view of the whole system as it will be built. You need to try various combinations of components, trading off this for that in the functional boundaries and interfaces, in the cost and performance. You cannot do this if you start building incrementally before looking at architecture. This is true of the enterprise as well as most solutions. Only simple systems can evade this rule.

It does not matter if you build or buy the components, the same factors apply.

2.10 DRAWING THE BOX, JUNE 22, 2014

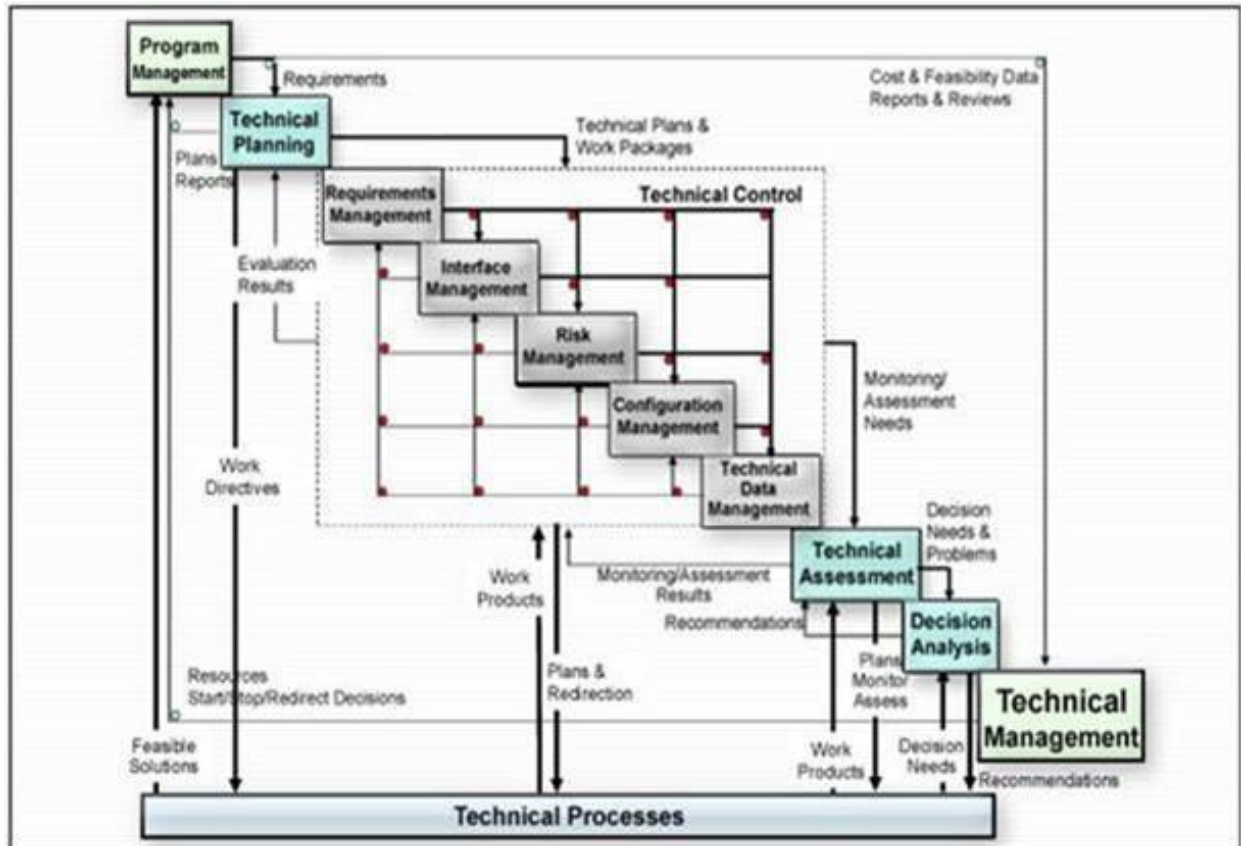


Image from US Government training Documents, DoD.

A fundamental concept in system engineering has often been ignored. This involves drawing a box around a certain set of functions or components and defining that as a system or subsystem. Drawing the box is a central issue in system engineering.

In traditional science, systems were “closed”, assumed to have nothing entering or leaving. Ludwig Von Bertalanffy nearly received a Nobel Prize for General Systems Theory, describing the behavior of an “open system”. An open system has interfaces which exchange well defined energy or material with the outside world.

In testing we have “black boxes”, “clear boxes” (sometimes termed white boxes) and “gray boxes”. In a black box, you understand the boundary, the functions and the interfaces but you have no means to see inside or test inside the box. In a clear box all internal functions are understood and accessible. In a gray box, some intermediate state exists. Black box testing **MUST** occur at the interface.

In general systems theory Von Bertalanffy stressed that Newtonian subdivision was not enough to understand systems. You could divide a thing, such as a human body, into parts but you would not

understand it. You must also understand the behavior and interactions between the parts. This interaction and its understanding would occur at the interfaces.

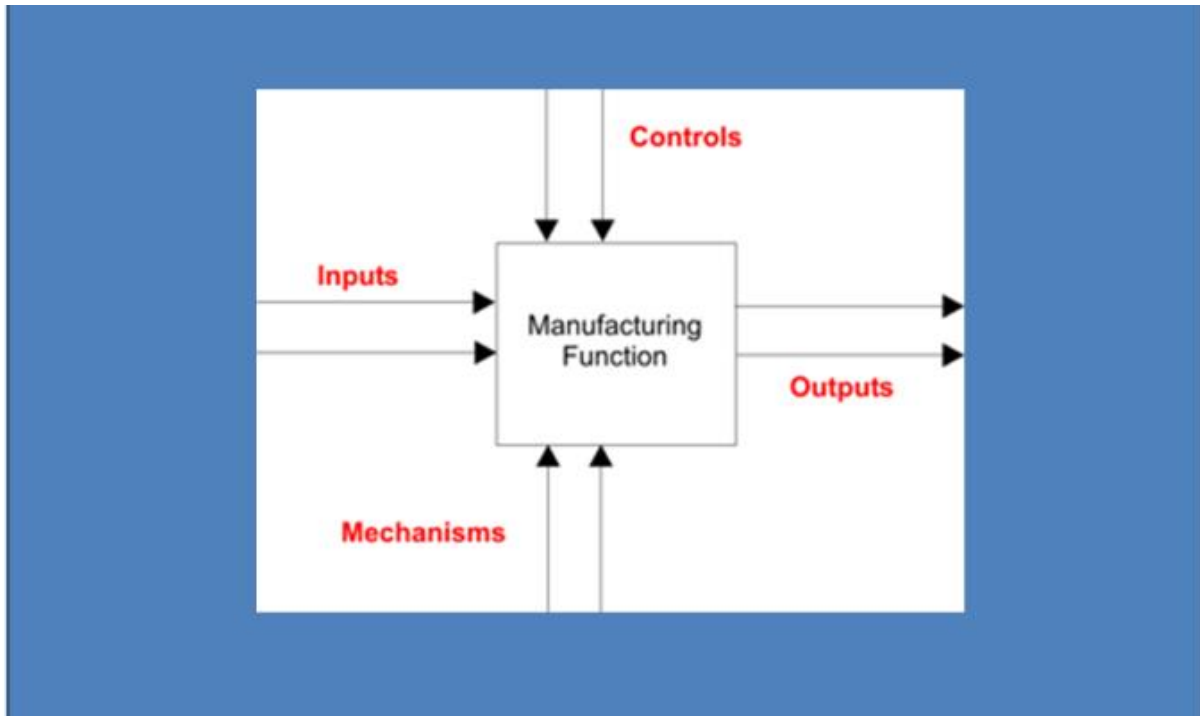
In engineering the identification and management of components in a design and their interfaces is crucial. Interface specifications are common. Allocation of functional requirements to components is common. It is fundamental in designing systems to understand where to draw the box and how to specify the interfaces. Standardization of components and functions and interfaces are the basis of production engineering for many classes of system.

If the box is drawn poorly interfaces become more numerous or more complex or more varied. If the box is drawn well interfaces simplify or become fewer. By choosing well the item being built may be assembled in parts, the parts tested independently, and parts may be reused or replaced for maintenance easily.

The next time someone insists to you that System Engineering is about the life-cycle model alone, I encourage you to ask them about General System Theory and Drawing the Box.

This entry was originally posted on my now defunct blog site in [Systems](#) on [5February2013](#).

2.11 DRAWING BOXES, DECEMBER 19, 2014



Much of system engineering and architecture can be seen clearly through the process of drawing boxes. People get all confounded creating excess complexities around this simple topic. It all goes back to a guy named Ludwig Von Bertalanffy and General Systems Theory.

Take any problem that you want to solve. Draw a big box to represent the solution to your problem. Draw inputs coming in from the left, and outputs leaving the right. Draw relevant controls like laws, policies or relevant standards coming in from above, and tools, budgets or other supporting mechanisms coming in from the bottom. This is how you start.

If you use the methods here and cannot solve the problem, you have probably defined the problem badly. While some problems simply cannot be solved for other reasons, most often a redefinition (re-framing) of the problem will lead to a solution. Some problems are said to be so complex and indescribable that they cannot be solved, and various groups debate these, but in most cases these can simply be redefined as well to produce solution.

If your inputs are bulk lead and your outputs bars of gold, you may have described a problem that is hard to solve. Re-define it.

Inside your big box draw smaller boxes using the same conventions (ICOM). If you draw a box that you will not need to look inside of (perhaps because it can be procured off-the-shelf), it is a "black box". If you will further decompose it (perhaps on another sheet) it is a "clear" or "white box". If you can only partially control or see the insides it is a "grey box".

By choosing building blocks that you have used before and know to work you can simplify the solution you describe. By breaking the problem into stages, functions or parts you can simplify solution as well. Very complex problems can be solved by breaking them down into simpler problems.

If you build up a larger solution from smaller parts, remember to group highly interconnected parts together- and have as few lines (interfaces) cross the edge of the larger solution. These rules for grouping things are well established, and you can find further discussion of them in basic system engineering texts.

The method will work for organizational structures or processes, manufacturing, electronics, mechanical devices and a wide range of other things. The application of this method to business process is called IDEF 0 and was a standard (now withdrawn) called FIPS-183 (and is still useful).

Now a set of things (depicted by boxes) and relationships between them (depicted by lines) is called an architecture. See IEEE or ISO standards for that. Your little drawing is an architecture. The process of breaking down a big problem into smaller problems and creating a solution is called system engineering. See INCOSE for that.

If you design or write software you may use Object Oriented Design and Analysis. Can OODA be extended outside of software to real physical objects? Sure! ICOM is more-or-less just that. (Note: Block diagramming techniques like ICOM have been used to produce CPU chips with trillions of transistors and many millions of gates, computers containing hundreds or thousands of chips, and networks containing thousands of computers- for example.)

While you can now go forth to debate endlessly with people who like to debate endlessly, you will also have a simple model to fall back on. The use of ICOM and IDEF was simply handy in this illustration. It is not required. It's all pretty easy to describe and use, really.

Two examples show success and failure in using such models. The SABRE system required the construction of an artificial construct, an abstraction, to simplify and unify its architecture. It went on to unify air travel and ticketing across all airlines in the 1970s. The DoD BEA (Business Enterprise Architecture) ignored the rules requiring locality of interfaces, producing models with boxes having hundreds of inputs and outputs, and this proved unworkable. It had to be redrawn, reanalyzed, or as the OODA folks might say refactored to be usable.

2.12 MANAGING INTERFACES, FEBRUARY 13, 2015



If you are performing solution architecture or integration, one of the most important functions you can perform is to manage the interfaces between distinct systems. These systems will probably have been developed separately, may be provided by different vendors, some may be "off-the-shelf", and some may be legacy. You must perform this function if you use MOM, EAI, ESB, SOA or messaging appliances, although the terminology may change slightly. Not a bit of this changes for the cloud in most cases.

Let's examine, for a moment, the commonly used and important aspects of managing such interfaces:

The Interface Control Document (ICD) is an agreement between the two parties managing distinct systems. It contains all technical details of interface at all 7 layers of the OSI model, or all of those applicable. It is an engineering document. It might even contain pin-outs of connectors where required. The ICD should not contain administrative detail, or it may require extensive signatures each time a technical detail changes.

The Interconnection Security Agreement (ISA) is agreement between two parties to exchange data. It identifies controls on the use of data by the receiving party passed down from the transmitting party, which allows sharing of data while maintaining legal or policy compliance. It also states the scope of data authorized for exchange. This document should exist whenever data crosses a security control boundary, such as between two companies. An ISA may take weeks or months to negotiate in complex legal environments. The ISA should not contain technical detail or renegotiation and new signatures may be required for minor technical interface changes.

The Service Level Agreement (SLA) defines the level of service, up time, response time, throughput and similar aspects of an interface. It should also not contain technical detail. Hint: If you desire to reduce the number of documents look to merge the SLA and the ISA (not the ICD).

The system or unit specification is commonly used when an interface or system may be reused with many other partner systems interfacing to it. It may specify a full range of features for an interface where any instance of use may not use all of the range of features specified. Such a specification does not replace the other documents listed, which may also be required. In a SOA service interfaces may be specified using WSDL, SOAP etc. and kept in a catalog, which is all specification. The so called "API movement" in web services is about specifications and reuse.

A system diagram (various names) should be kept for any complex integrated system environment (System of Systems). It should depict and identify systems and the interfaces between them, usually accompanied by lists of interface details and lists of system details. Matrices of interconnection may also be used. A separate diagram of network details is often also advisable to support maintenance.

The Configuration Control Board (CCB) approves and directs changes to the systems, the interfaces, and these documents.

The most common mistake is to try to combine these documents in odd ways. The normal set of documents is time proven to avoid administrative and technical trouble. Not knowing how to manage interfaces has been a factor in many large system implementation failures, some in the news over the past few years. The process is not agile in any real way, and is typically not managed by programmers within a sprint.

2.13 FUNCTIONAL, VIRTUAL AND PHYSICAL ARCHITECTURE, SEP 1, 2015



Today, and for many years, there have been three kinds of solution architecture. The solution architect is commonly called on to produce any and all of these as needed. Any or all of these may contain multiple artifacts to describe platforms, interfaces, functions, users, processes, material or information flows.

Functional Architecture

Before you have your components selected, before the design is finalized, you may take the functions of the solution and assign them to preliminary or abstract blocks. You may then detail the interconnection of these blocks with notional interconnection.

Some call this "logical architecture", but I dislike the term as it is ambiguous and may also be used for your "virtual architecture". It may be used long after the solutions real physical components have been specified, or it may be refined over time and used as the basis of description for a "design pattern" or "reference architecture".

Physical Architecture

When you select real components and allocate the functions to them, based on their individual capabilities and lack thereof, you create a physical architecture. This should then be accompanied by real interfaces with real interface details. Model and revision are important in physical architecture, as may be real weight, real temperature constraints, real power usage, and the rest.

Virtual Architecture

The virtual architecture does not show real physical platforms or real interfaces, it shows the virtual equivalents. If the real server uses virtualization to support five virtual servers, the five will appear in the virtual architecture and the one in the physical architecture. Virtualized interfaces, interfaces resident in virtual machines, , as configured, will appear in the virtual architecture.

Artifacts

All three are depicted with the usual engineering artifacts. Lists (schedules) will identify parts, functions, services, and anything to be enumerated. Matrices will relate one list to another. Drawings will depict the information in a way that takes advantage of the visual acuity of the engineer or technician involved in construction or maintenance, or of the manager if capable of grasping such visual information (some people are not and this is not a key requirement for a manager). Documents may describe some aspects of the system represented, such as use cases.

2.14 SYSTEM ARCHITECTURE QUALITY, FEBRUARY 21, 2015



If you think about it for a moment, not all architecture is good architecture. Some is junk and some is pure gold. It's very odd, but this topic is not taught much, is not covered by frameworks, and is notably absent from professional dialog. Don't confuse this with EA process maturity, designed to improve EA to produce, ostensibly, consistently better architectures. I am talking about the quality of any one architecture, as delivered.

Measuring Solution Architecture

This is not hard to do. Just examine the system that would be produced, or was produced, and see how it stacks up against what the business processes, the operational mission, and the environment require. Measure it. Test it. Evaluate it. If you did a lousy job, try to improve that next time, or even in the junk you built.

Comparing Architecture

You can have different solutions to the same problem. I did a group thesis at National University with Paul Agosta and Chuck Frawley (great folks, very smart), and part of our thesis was that you could compare architectures. You could look at the systems that would or did result from the architecture, and examine them by various criterion, then compare. You can say "this architecture is better for this purpose, and terrible for that one". We demonstrated doing that.

Some time ago large systems for the US Federal Government were procured in competition where the architecture was a major differentiator causing differences in cost and effectiveness. Now much of procurement just buys bodies, and the effectiveness of what they build is controlled and limited by government management. Some of what we lost in that is examination of the architecture and whether the resulting system is good or bad.

Building a Better Architecture

In that thesis I mentioned we took three systems and compared them, then extracted the best features of each. We then built a new hybrid architecture using those best features. The result was, on paper (we had no money to build it) far better. It would have seriously improved DHS organizational performance in counter-terrorism and natural disaster response. Better architecture can have serious impact.

Why Junk Architecture

Junk solution architecture can be caused by several factors:

- Poor or unqualified architects is a big factor. Everybody wants to be the architect, few are any good.
- No time for architecture.
- No money for architecture.
- Politics is used instead of science, engineering and empiricism.

Architecture ROI

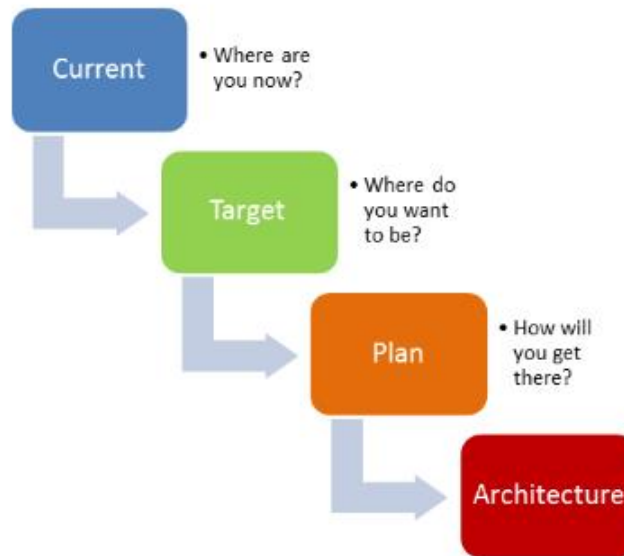
If you produce junk architecture, or no architecture, you can waste millions. On one program I saw 3/4 billion wasted. On the other hand a good architecture can create organizational transformation and massive organizational performance improvements. ROI for architecture is related to quality.

Conclusion

Before saying architecture has no ROI, or is a waste of time, examine if your data is based on junk architecture. In recent years with the massive flood of unqualified architects, or software developers posing as being competent to design the complete system (*enterprise architects and system engineers are specifically trained or educated to merge technology and operations/business*), we have had a huge number of poor impact systems and failures. Look at VA data exchange, or the Obamacare site, or the current ongoing debacle at USCIS (yes my opinions, but check GAO as well). Good architecture has very high ROI, and poor architecture has expensive consequences.

Stop using poor architects and then claiming all architecture is flawed!
When you buy a cheap architect to save tens of thousands and the losses climb to millions we are all out here laughing at you! Once you have good architects, let them do their jobs!

2.15 WEBSITE SOLUTION METHODOLOGY, MARCH 22, 2015



Web companies seek architects specific to the Web who understand the solution processes and methods used for websites. Yet the process used for a website engagement can be derived from the general solutions architecture process for strategic solutions (mission relevant) in an hour or two by any competent architect. Simply because the recruiter or hiring manager lacks the ability to generalize, synthesize or apply architecture methods, they make assumptions that architects cannot. By doing so they may be rejecting more qualified architects for less qualified architects.

To demonstrate the point, I will take the Clover™ method and tailor it to produce a commercially viable and superior website methodology, right here, right now. (See Clover™ here: <https://www.linkedin.com/pulse/lover-method-matthew-kern?trk=mp-reader-card>)

STEP 1) CURRENT STATE ACTIVITIES:

- Inventory Website(s): Identify scope, roles, personas, supported processes or functions, content, technologies, governance, navigation, color schemes, branding
- Perform Website SWOT Analysis
- Identify and Interview/survey Stakeholders concerning perceptions of website
- Gather any other available measures of effectiveness

STEP 2) TARGET STATE ACTIVITIES:

- Produce concept of operations, scenarios, user stories, new performance targets
- Identify new or changed scope, roles, personas, supported processes or functions, content, technologies, governance, navigation, color schemes, branding
- Produce alternative concepts, mock-ups, and have customer stakeholders assess them

STEP 3) PLANNING ACTIVITIES:

- Estimate
- Produce Project Plan
- Produce Functional and Derived Requirements
- Plan/Produce Procurement Package(s) as required

STEP 4) ARCHITECTURE ACTIVITIES:

- Produce User Interface (Wireframes), style guide, style sheets, branding
- Map scope, roles, personas, supported processes or functions, content, technologies, governance, navigation, color schemes, branding into coherent approach
- Produce Prototype and Evaluate
- Produce specification package.
- Then you transfer the resulting procurement and specification packages to the implementer.

I have performed some large website work myself, and assisted others in the same. The process above will work fine, and can be tailored a bit more to your needs. This process looks much like the process for any other solution. There is not much difference.

Most architects will recognize the first 3 steps as the strategic planning process, focused down to a solution scope. The last is architecture, designing or choosing the solution. Zachman, DODAF and TOGAF are all applicable to solutions work, by the way.

Recruiters, you are focusing on the wrong skills. Note the relative absence of system administration skills required in architecture. However there are a great range of other skills required above. Only a few are specific to websites alone. Architecture is a discipline and has skills associated with it, it is not your job to redefine them. You do not have the experience and knowledge to do so.

2.16 ABOUT ENTERPRISE SOFTWARE, DECEMBER 24, 2014



I worked for a time at Advanced Technology Incorporated in Reston. It was smashed into PRC by Emhart, maker of garden hoses. Black and Decker, who thought such a mashed culture was sensible and synergistic, bought the combination. For a time we had some Mormon guy who had never worked in software as a CEO- and he turned the logo Kelly Green. After I had left, the Borg scooped it up and assimilated it into NGIT at a discount.

Big Custom Software

In those years we produced big software. Often this was logistics software. It would track who ordered what, who could order what, and shipment of “what” to “where”. This software was closely tied to the business processes and methods of an organization. A new customer would require changes to the big database model that was the world view of the organization, and the software. The forms that allowed entry, update or viewing of the software would change to accommodate the roles as implemented by the organization.

In the data models of this software there would be sections that were mostly used by this organization or business function, and other sections used by other functions. The different business functions or organizations would define modules.

Not just logistics software, but job tracking, maintenance tracking and other functions might be included. For a time, storing the blueprints to support the logistics was important, so the plant could be maintained. Parts of the company sold software to nuclear power, others to the military, others to communications plant owners. Sometimes manufacturers would buy it as MRP (Materials Resource Planning), and at other times as collaborative engineering software.

In all cases the database model, the forms, the business rules and the reports had to be tweaked for the next customer. Each did business differently. This diversity in process created different operational advantages in different customer companies.

ERP

In that business we watched the slow motion train wreck that became ERP. Vendors would sell ERP as product. ERP included modules for all the big custom software types we once delivered. The sales cycle would allow the customer to lie to themselves, saying that they would be the ones who required fewer data model changes, fewer changed forms, fewer new reports. Then the vendors, not we services companies, would make the changes after the sale.

For years the stories built up. The blown budgets and slipped schedules were legendary in size and scope. We watched in horror as our favorite customers were mauled by the mythology that these enterprise systems were products that could be reused. Some left our business to help theirs.

Today even more big product names and categories are purchased together to be integrated into a large enterprise application, automating what are often the core business functions unique to one customer. Yet still the headlines are full of cost overrun and missed deadlines.

Enterprise software has many forms, automates the workflow of a business process, has a large central data model, and must be customized for the operations of each customer. Many people say I know about "enterprise software", and this is what I know.

Different

I also know that smashing together many large COTS applications, as is, is a different approach. "Integration" is what that is called. The two do not mix. Each has its limits. Neither is really "agile development" in the standard sense. One focuses on existing integration points and APIs, another is more like CASE/Method- if you want to succeed.

Example

Because I know these things about enterprise software development patterns I can sometimes predict disaster. For example:

- <http://www.gao.gov/products/GAO-07-1013R>
- <http://www.gao.gov/products/GAO-12-66>
- <http://www.fierceregovernmentit.com/story/uscis-transformation-behind-schedule-over-budget/2011-11-23>
- <http://www.nextgov.com/defense/2012/12/dhs-recompete-online-immigration-program-crisis/60144/>

In this example the vendor proposed use of 20 COTS packages, then ripping these apart to interconnect them via SOA and an ESB. New software would be supplied and compiled into the products to interconnect. I have referred to this approach elsewhere as MOTS.

Because the number of COTS products was too high, and rewriting so many applications to merge so many semantic paradigms is difficult, little progress was made using the initial approach. (When

you use COTS integration remember to try to use the APIs and "integration points" built into the ordinary COTS product, and avoid massive rewriting.)

To augment the initial approach additional development of large scale enterprise applications (large central database, many forms, automates processes) was added. These additional efforts were redundant with the initial work, and expensive. The new efforts were complicated by adding "agile" processes that reduce overall architecture to tactical reuse of software elements. This architecture approach is not sufficient for large scale enterprise applications, and has not achieved "alignment" or "organizational performance improvement". Specifically lower-level architecture must be driven by higher-level architecture, and ensured by governance.

http://www.unauthorizedprogress.com/images/EA_as_5_activities_2014.pdf

As detailed in the internal OIG report below, the new efforts have had limited effect. A small subset of total function was identified for a custom component system called ELIS. While enterprise architecture was identified as highly mature in the report, the management that achieved this has departed. The maturity level is also exaggerated, ignoring a lack of vertical architecture integration (not measured). The process analysis required to achieve transformational business improvements was not used to drive application function, and the levels of architecture were not integrated via governance and stage-gate reviews. Alignment was not achieved. Process performance improvements were not achieved. (See my other posts to understand the methods required to achieve these objectives.) (Note: The report incorrectly isolates these problems in infrastructure improvement, and infrastructure planning is a function of enterprise architecture.)

http://www.oig.dhs.gov/assets/Mgmt/2014/OIG_14-112_Jul14.pdf

Based on publicly available information I predict this will not end well, wasting significantly over \$1B of taxpayer funds.

Conclusion

Enterprise software is not like building a product. Applying product development methods has not yet made an operational impact in the case identified. It probably will not, as it lacks the correct corporate controls and analysis.

Enterprise software has many forms, automates the workflow of a business process, has a large central data model, and must be customized for the operations of each customer. Significant skill is required to make such efforts succeed.

2.17 THE VALUE OF ENTERPRISE SOFTWARE, AUGUST 3, 2014

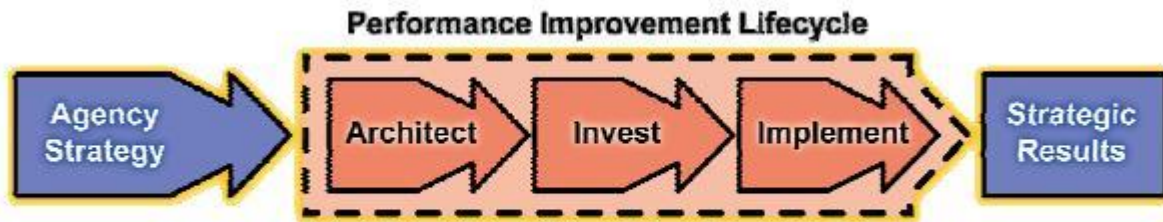


Figure 1-2: Performance Improvement Lifecycle

Source: FEA Practice Guidance, 2007, OMB

Measuring value in enterprise software is a bit different than measuring value in consumer grade software. This is not surprising, the enterprise environment is more complex and you serve a mission or commercial purpose in an enterprise.

Test and Evaluation

Let me introduce a concept here for you to consider. In system engineering we say that testing verifies that you have built the thing right. You test that the thing meets the specifications.

However in evaluation, they say, you evaluate that you have built the right thing. You see if the specifications and concept are effective in real use. My point here relates more to evaluation than testing.

The Purpose of Software

Enterprise software exists to improve the business process it supports. Processes are typically improved in regard to throughput, cost or quality. If the software automates some step so that it need not be performed by humans, it has value by reducing cost and maybe improving quality or throughput.

Uninformed Users

Sometimes the user of a bit of software is not aware of the new process being implemented. This happens often in transformation, where that user role is executing an obsolete process slated to be changed. Their old job may be greatly modified or may have completely disappeared in the new business process.

I may not want my developers implementing an excellent bit of software that is magnificently helpful in automating last year's process. I may want my developers focusing on the new process in a written specification, a process never executed by users to this date.

Measuring Value

In enterprise software there may be performance measures tied to the process being improved. Changes in those performance measures may be the primary determination of the value in developed software. Delivering code that works perfectly but implements that old obsolete process may not be a relevant indicator of progress.

Principles behind the Agile Manifesto

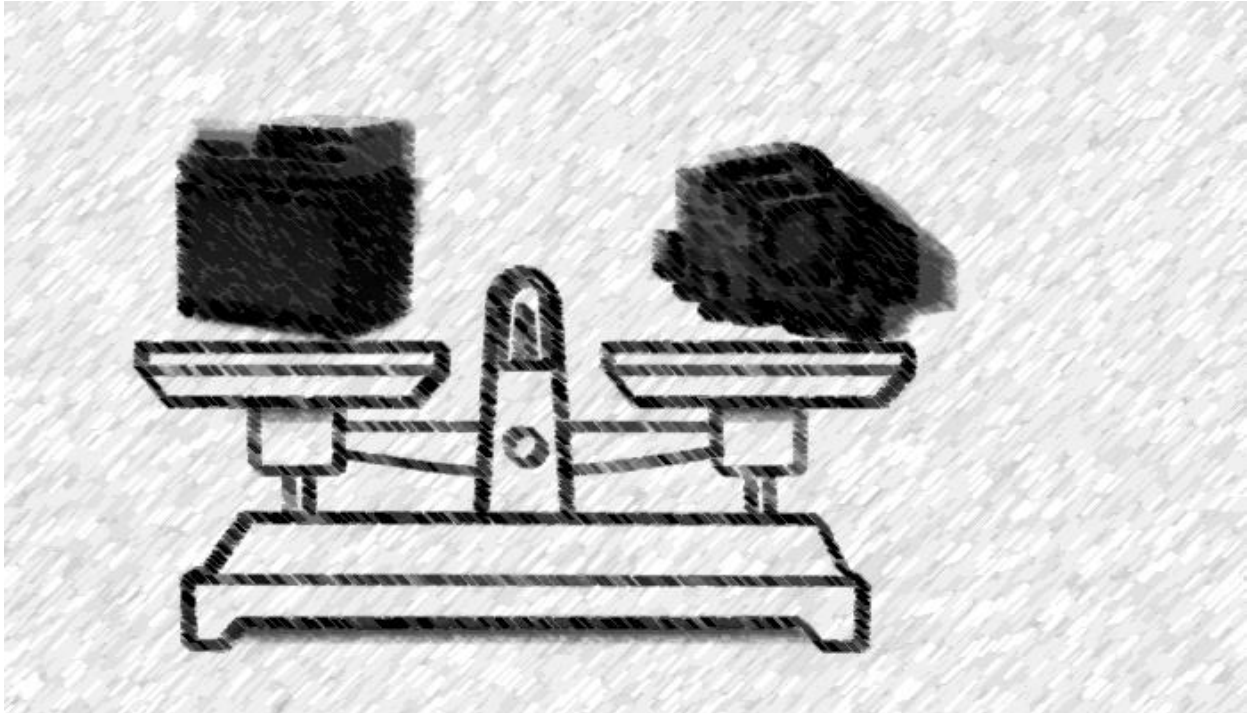
In the web page so named I find two significant errors of thinking. First it states "Working software is the primary measure of progress". In the enterprise a large heap of expensive and exquisitely working software that implements an obsolete and ineffective process is not progress.

Second it states "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. "This may be true for trivial development, but it is less true for complex new business processes that have never been executed. It is also less true for back-end development with complex data structures or transformations. In those cases written specifications and process diagrams may be far more effective and far more efficient.

Conclusion

The value of enterprise software is often measured by its effect on organizational performance in the enterprise. Creation of effective software for new processes may require complex specification and documentation. An Agile approach may improve software development, but may still produce inferior value.

2.18 CENTRALIZATION VS DECENTRALIZATION, AUG 19, 2015



We often read that we are in the midst of a wave of centralization of computing. In decentralization there are many computers, spread around, independently hosting applications. In centralization fewer computers in central locations do all or most of the computing. I will offer some conjecture.

Some History

In the dawn of the computer industry there were few computers, and they were expensive. Some could perform only specialized tasks, but the general purpose computer quickly emerged as superior. Such general computers could, at first, run one program for one user at any time. Systems to automate running job after job, faster and better, emerged and were then followed by systems that allowed multiple jobs (applications) for multiple users to operate in apparent simultaneity, connected to individual user terminals replacing older punched card stations.

The large central computer was engineered for throughput, maximizing the amount of processing performed by the processor with little or no regard to responding to a user or event in a predictable or rapid way. These "mainframes" were built with "channels" to offload processing of input and output operations to other "controllers" to maximize throughput. Operations like moving a file or searching for a record in a database were performed without slowing the central processor.

However two innovations eroded this paradigm of computing hardware and software design. The first, minicomputers, created systems more responsive to user input and more applicable to functions such as industrial process control. When an input occurred the output was available more rapidly or more predictably than with the "mainframe". (These had a different internal architecture, again, with independent "busses" for input and output controllers. These busses differed from mainframe "channels" allowing different devices to take control of different

operations rather than central arbitration. This decentralization improved responsiveness, but often not throughput.) Replacing a mainframe with several minicomputers created a more responsive computing environment.

Also terminals became smarter. The "microprocessor" in the typical terminal became more capable. In addition to the keyboard, optional local printer and video display of the terminal, the "personal computer" added local disk and the ability to run local programs without recourse to a centralized "mainframe" or "minicomputer". The "microcomputer" had a simplified single bus structure for input and output. Input and output were reduced, but responsiveness was now maximized.

Physics

The physics of computing dictated these changes. The transfer of information between two discreet electronic devices, transistors or chips, at any scale of integration, is slower than the transfer of information within a device. Furthermore distance increases transfer time and reduces speed of information transfer. For speed to increase and cost to decrease individual units had to become physically smaller but functionally larger, more capable, incorporating more and more transistors and gates on an "integrated circuit". We progressed from small scale integration to medium scale integration to large scale integration to very large scale integration to extremely large scale integration to unnamed regions of device density on a single chip.

This incessant progress of device size reduction and integrated circuit functional increase became embodied in [Gordon Moore's Law](#), which has dominated computing progress for 50 years. However as devices become ever smaller their predictability of operation is less modeled by the aggregate physics of the macroscopic world and is more accurately modeled by quantum physics. Further, external events such as stray particles have an ever increasing probability of changing the output of a device, a stray neutrino might change a single digit in your bank balance, in any significance from pennies to millions. Moving from classical computing based on [George Boole's](#) logic to quantum computing is inevitable, however it seems uncertain that all computers will switch paradigms. It is perhaps easier to program with classical logic.

Simultaneously the single layer of devices in the chip are a barrier. Efforts to create chips with two or more layers of active devices have recurred over the years. Similarly, stacking chips within a package is a recurring area of research. The limits on these efforts are usually cooling, dissipation of the heat created. This theme is universal as computing density increases. Moore's law must perhaps end due to practical considerations of thermal transfer even if devices become single atoms.

Decentralization

To achieve more computation we have adopted parallelism many times in the history of computing. Processing units inside CPUs are parallel, processors are often parallel, and computers are linked in parallel to create supercomputers. This works for the majority of problems that we employ computers to solve but not for a very few. Some small set of problems cannot be solved in parallel, and solution must be sequential. Mathematics and computational theory tells us which are which. Some of those sequential problems may be conquered by quantum methods, some perhaps not. Excluding that small set, all the rest are subject to parallelism, and to decentralization.

Computers are now embedded into almost everything. My car has more than one. My cell-phone has one or more. My home phone, a vanishing feature of daily life, has one in each receiver and one in the base station. My music system has one. My television system has several. Many of my appliances and some of my individual lights have computers embedded in them. Computers are everywhere, described and predicted years ago as "[ubiquitous computing](#)". This trend has only started, and will continue to increase over the foreseeable future. These embedded processors are not being centralized.

Decentralization is alive and well, and growing prodigiously. It is perhaps best known as "The Internet of Things" or "IoT".

Centralization

Why bother to have your own data center? The expense of managing your own computers is large, so outsource it and save money! This was the original value proposition for "cloud computing". The core technology involved, virtualization, began on those mainframe operating systems back in history, allowing the jobs (applications) of individual users to be resourced, scheduled, executed and billed individually on shared hardware. Costs were reduced drastically. Instead of many identical mainframe processors in a network (SNA) we now have many microcomputer servers in a network (TCP/IP). Unfortunately the centralization of processing, "The Cloud", has arguably failed to produce cost savings - so far.

But other advantages are being marketed as they are thought up. Some will "stick", "get traction" in a marketing sense. The scale of this centralization effort is staggering, as the scale of modern computing is staggering. The end game is to produce a small set of vendors who sell computing as a commodity. A single vendor is a monopoly, and we avoid that in democratized capitalism as a sort of commercial fascism, dictating price on the supply side. We want lower price. So we will have at least three vendors, I think. My friend, former CTO and systems architect [Kevin Lorenz](#) argues that it will be two big players and several niche players. AWS™ will dominate with majority market share, Microsoft Azure™ will follow with a minority share of the market Kevin predicts, based on present trajectories. I find his logic persuasive, which he would agree holds only if present trajectories are not altered. (When the US Federal Government realizes it is creating yet another oligopoly with no price advantage, what will happen to present government cloud promotion efforts?)

None of this changes the trend toward simultaneous decentralization, and the momentum of that (Internet of Things). Centralization will not work for cell phones, cars and such. It seems to be targeted at enterprise computing, and decentralization at personal computing.

Enterprise Computing

The key aspect of the difference between applications being centralized and those being decentralized is as predictable as the next DC heat wave. It is the key central resource. The database is at the core of the enterprise application, and the database is difficult to distribute. Physics again dictates the agenda.

Any database, of any size, may be distributed. For analytical applications (OLAP) this is easier than for transactional applications (OLTP). Distribution of a database involved in transactions requires protocols between the servers such as 2 phase commit and 3 phase commit. These protocols are

widely known, and were even embedded into the ISO-OSI communications model years ago- then ignored. Recently we have products like Hadoop to manage database distribution. When two database servers are separated by some distance, the speed of communications between them slows and the transaction rate slows, for many applications limiting the database to existence in a single data center or two in the same city.

Enterprise applications, often complex with many subsystems like accounting or inventory or HR of a size dwarfing the entire scope of other classes of applications, are usually collocated with their data, improving access speed. This argues for centralization. Tricks may be used to partition data onto many servers, cutting the single data model into parts and fragmenting its world paradigm. MongoDB™ is such a platform. You can find loopholes in the physics in this way, sometimes, at some expense in areas such as atomicity or data integrity. Otherwise one big database with one central website for the one big application seems like the winning architecture, as dictated by physics.

But applications too are subject to economics and regulation. ERP vendors such as [SAP](#) have tried for years to capture all users everywhere on a single central platform, under one brand, eliminating the competition. Like nearly all other efforts to monopolize supply of enterprise applications, SAP failed to capture 100% of the market. There are always competitors. We like competitors and dislike monopolies.

Vertical Integration

The end game of cloud vendors will be vertical integration. This market phase is inevitable after all the new profits from centralization of platform are captured. In this phase the few cloud market vendors will pick the few favored enterprise applications, and alliances will be formed. Companies will merge or acquire. A shakeout of applications will occur. Eventually there will be only a few offerings of enterprise software.

This ignores the need for mission specific enterprise software as a differentiator at specific end customers, allowing those customers to perform better than their peers. To date the West Coast driven cloud vendors are clueless about how to build such mission enhancing software, but they currently seek to eliminate or control all those pesky little competitors who produce such custom applications.

Security

Such market forces introduce many security risks to company operations, data privacy, and operational freedom. Most of these issues have not been addressed. Centralization is not the only source of threat, decentralization offers a vast surface for penetration and exploitation. Lastly, data is most secure when it is in your physical control.

Conclusion

We are currently undergoing both centralization and decentralization of computing. Applications are being split between those best centralized and those best distributed. Current trends have implications that are not all favorable, and these efforts may stop long before their predictable market end-points.

I might project, looking one or two steps into the chess board that vendors of enterprise software might be best served by moving to own their own small cloud where their applications reside sold as SaaS. Groups of such software vendors might band together in groups to share the costs of ownership. Vendors of custom enterprise software might form or join similar alliances. Such positioning would be best if the cloud platform was not either MS Azure™ or AWS™ who will eventually be pressured to eliminate you in favor of a favored vertical partner.

If you are a customer organization that does not produce software for revenue, and if you have worked very hard to develop a custom bit of software to improve your operations, you may be best served hosting it in a private cloud on your very own computers under your own control. This is especially true if the data is sensitive or classified. This will remain true, unless and until, cloud computing produces the initially promised cost advantages.

2.19 THE VERY LARGEST DISTRIBUTED SYSTEMS, SEPTEMBER 14, 2014



What are the largest distributed systems? What technology are they based on? How are they organized? The answers may surprise you. (Note: We will exclude centralized, homogeneous, co-located systems as not particularly "distributed".)

Big Distributed Systems

Probably the largest distributed systems on the planet are these:

- The world-wide banking system;
- The Visa credit card network;
- Stock exchanges.

Each is a large system of relatively autonomous information systems, coordinating their actions on a large scale.

Asynchronous Messaging

Each uses asynchronous messaging and LRTS to coordinate between independent systems. Within each system (not between), less scale-able SOA like or hierarchical mechanisms may be used, depending on which participant system you may be looking at.

Scope of Messaging

One important organizing principle in these systems, is that the core set of transactions in each are few. This principle was rediscovered by USAF and MITRE working to apply new organizational approaches to C2 (Command and Control) in the CoT program (Cursor on Target). The more participants of diverse interests are included, the fewer topics they have in common. It seems obvious, but it took MITRE and USAF to articulate it.

Grid

In each case the system is organized in a grid, and not in a hierarchy. As the number of nodes grows the need for reliability and availability forces the removal of single points of failure and adoption of a more robust, more fault tolerant grid topology.

Security

In no case are these systems moving data across the open Internet, all move data across private networks. In all cases there is strong encryption, not involving PKI.

Network Appliances

All of these use hardware accelerated network appliances. None rely on rigging general purpose servers to act as interconnecting nodes; thus, performance is orders of magnitude greater and cost to performance ratios are lower for these nodes. In fact the very low message latency and high throughput on these hardware accelerated appliances enabled the wave of HFT (High Frequency Trading) applications that were not possible before such hardware.

Choreography, not Orchestration

Each party knows the protocols and sequences in communications. No central organizing, directing or sequencing function is used. Standards based protocols are used. SOA Governance like mechanisms are not used.

Candidate Topics

One odd finding is that the number and range of such large scale organizing topics seems to be small. Having explored this a bit, I have identified a few and rejected many:

- Trade, buying and selling, exemplified by EDI protocols and their XML adaptations.
- Situation awareness and C2, as exemplified by the USAF Cursor on Target protocol. This includes the so called "Internet of Things".
- ICAM or IDAM, the management of identity and access for computer security.

Implications

There is a scale so large that the typical hierarchical SOA mechanisms are not effective. If you need to get very big, there is a different body of technology to implement it. This conclusion is supported by the engineering reliability equation, which I have written about elsewhere In these volumes.

SECTION 3: TOPICS:

Section 3: Topics:	75
3.1 Why Federal IT Projects Fail, Sep 16, 2015	76
3.2 COTS/GOTS vs MOTS vs Custom Development, July 4, 2014	78
3.3 The Albino Rhino Hunt, June 13, 2015	81
3.4 Is the Cloud New, June 29, 2015	83
3.5 Cloud Service Categories, August 23, 201	85
3.6 The Cloud Paradox, SEPTEMBER 28, 2014	87
3.7 Security vs Cloud, August 23, 2014	89
3.8 Essential Cloud Architecture for Customers, March 21, 2015	91
3.9 The Cloud Killer, June 30, 2015	96
3.10 What is Big Data & Why No-SQL?, August 23, 2014	99
3.11 Basic Truth: OLTP and OLAP, June 1, 2014	103
3.12 Fun with Database Servers, Sep 4, 2015	105
3.13 Building Large Transactional Application Suites, July 15, 2014	108
3.13 Solution Architecture Framework, Never Posted	111
3.14 System Development Lifecycle, Never Posted	114

Posts 3.1 to 3.3 describe some illusions that keep us from success. Posts 3.4 to 3.9 concern ðthe cloudö. Posts 3.10 to 3.13 deal with database applications. The last two posts discuss choosing the framework and SDLC for your efforts.

Questions for Section 3:

1. Will ðthe cloudö dominate the next 20 years?
2. I have said that data analytics will outlast the ðNo SQLö wave and ðBig Dataö tools. What ordinary relational databases have incorporated some of these features so far? Am I wrong?
3. Which SDLC do you prefer and why? What makes an SDLC ðgoodö for your purposes?
4. Which framework do you prefer and why? What characteristics do you need in your framework?
5. Will knowing the influences linked to failure help you avoid it?

3.1 WHY FEDERAL IT PROJECTS FAIL, SEP 16, 2015



I will try my hand at a better top ten list for why Federal Government IT projects fail. I saw one recently, and I think I can do better. I will do it free-style, without a net, for this pass. I believe I may have a better perspective than most vendors, having worked for the government customer for many years. In the original post I asked for comments on how close the list came to the reader's experience (How did I do?)

1. **Vendor Hype.** Often vendors oversell some solution or method, making excess claims. They do this in a formal proposals, with great credibility. They may do it with the backing of pundits as an industry trend. Unfortunately, in practice, the solution does not deliver.
2. **No Real Organizational Benefit.** Often the acquisition was never thought through, and the performance of the organization (not the software) after implementation is no better. After spending a ton of money, the government folks who do the real work see that the change has been worthless. This happens when the stage gate system defining how the system will improve the organization is bypassed, and test or evaluation to check that that the operational link is working may also be bypassed. In many cases no business plan may have ever been written or reviewed. The system may succeed but the application may fail.
3. **Not Understanding Agile Limits.** Agile is great for small projects focused on human interface. Agile often fails for complex projects of large size focused on back-end function. This is true even with Agile extensions. Complex system development requires additional systems engineering, architecture, and testing/evaluation efforts. Applying Agile to the wrong systems has resulted in widespread and catastrophic failures.

4. **Confused COTS, GOTS, MOTS, Custom tradeoffs.** (aka Poor Buy vs Build Decision, or poor alternative analysis). You can buy an off-the-shelf system and change the business processes to fit it, or you can build a custom solution to fit your custom processes. Custom software is more expensive than off-the-shelf software, but costs of changing internal processes are often ignored. It is far more expensive and difficult to modify your off-the-shelf software to fit custom processes, more so than either custom or off-the-shelf alternatives. Risk skyrockets. This is poorly understood in many government agencies. Failures result.
5. **No Distinctions between Mission Systems and Support Systems.** Mission critical and mission central systems may require custom development, as often there is no one else doing that mission and no software to automate it. These systems, of direct import to the mission, have the most likelihood of increasing organizational performance. Support systems may as well be COTS, as they will not substantively improve organizational performance. By putting too much effort into support systems money required for real benefit is squandered, leaving insufficient funds to implement mission systems.
6. **No Logistics Train.** Once the solution is delivered, there is often no plan to support or maintain it. Costs skyrocket versus estimates. Funds run out. Failure occurs.
7. **Ignoring Non-Material solution.** Often the government agency could simply change the way it operates and buy nothing to increase performance. Vendor and political pressure may force a purchase anyway, so as to appear to be doing something. The results may prove immaterial. (This is a sub-case of item 2)
8. **Poor Project Management.** Often agencies ignore standard practice from PMI or decades of project management wisdom in favor of simplified methods. Stakeholders may be ignored. Scope may increase. Requirements may change too much. Risk may not be managed.
9. **Lousy Requirements.** This is a subset of item 7. Customers often get what they specify, even if they specified the wrong thing. Sometimes customers specify too little, and get random results or missing functions.
10. **Integration in a Can.** Many organizations think they can just leave integration as an afterthought, trusting SOA or EAI or ESB to fix everything. Integration takes work and analysis, specification of interfaces and testing. Improper effort may cause failure.
11. **Zombie Investments.** The Federal Government is not very good at canceling programs. Sometimes when a program goes awry, it remains active with a minimum of funds. At such funding levels it can never achieve objectives, and it fails against the original plan.

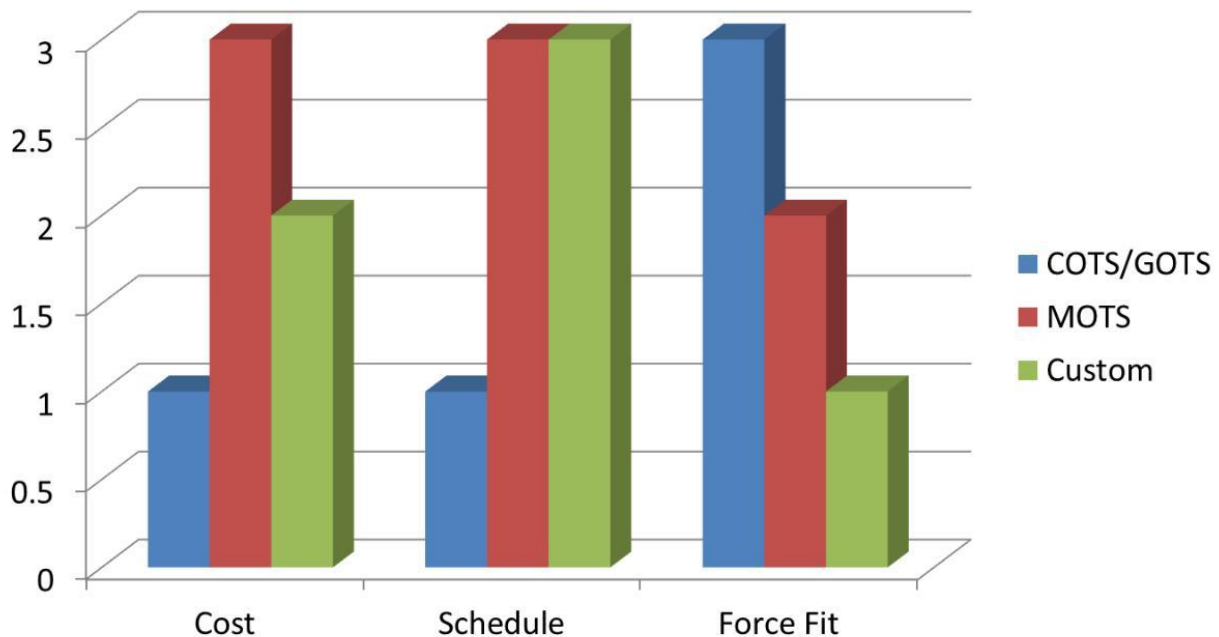
OOPS, I got 11 instead of ten. It was a quick try. (I asked for comments again here.)

COMMENTS CONTRIBUTIONS

Selected commenters (the ones from which I could discern a contribution) added the following to the list:

1. Gary Klepper added force of law to the list specifically for state projects;
2. Dale Chalfant added "Not doing things in the right order";
3. Dmytro Kurylovych added corruption and bureaucracy;
4. Robert Vane added functional domains (silos) as a problem.

3.2 COTS/GOTS VS MOTS VS CUSTOM DEVELOPMENT, JULY 4, 2014



Suppose there is some large and very complex application at some large and very complex organization. They need an all-encompassing environment to improve operations. (Large ERP systems are an example.) Two approaches to buying that application suite/environment are often identified and compared.

COTS

The first is buying that application suite or environment "off-the-shelf". You will attempt to buy the whole thing from ACME or BigSoftware.com. (Which are trademarks of their respective companies, no doubt). These work well, are proven, have few remaining errors. They can be installed and operational in a relatively rapid period of time. The cost is low, as the code-base is shared with many other users- both for development and ongoing maintenance. Unfortunately the software may make some assumptions about your operations and methods, and you might have to change practices to fit those assumptions.

GOTS by the way is a term used in government for reuse of applications whose code is owned by the government. For the sake of this discussion, it is like COTS.

Custom

The main alternative is custom development. You can have a large application custom built to fit your existing processes and operations. This is more expensive, but the quality can be far higher due to this fit with existing practice. Maintenance cost will also be high, as you share this custom code with no other users. It may take an extended period to develop such a custom application.

The diagram shows a comparison of these methods with the range just being high, medium or low in the category. No fine grain interpretation of the graph should be made.

MOTS

These two approaches are discussed as "acquisition approaches" in US Government Federal Procurement. A third has escaped notice. A very senior systems engineer at DHS (I won't mention his name lest someone misquote him as if it were a new DHS policy or something) termed it MOTS, or Modified Off-the-shelf. It has become very common.

MOTS involves buying several COTS packages and then attempting to change them with new custom code. Proposals say that such an approach will save money and produce customized application fit to your existing or desired business processes. That rarely occurs.

What actually often happens is that the MOTS code-base is sufficiently customized that maintenance costs skyrocket, as the fixes used by the total user community must be separately applied to, developed for, your highly customized version.

Also the set of developers that can produce within these mature complex production applications are expensive folks. The cost of development of new features often escalates.

The resulting changes do not provide infinitely close match to your business processes, as the paradigms and world model of the production applications constrain new features. Changes linger on forever, with users demanding fixes to make the software better fit actual processes.

Results

MOTS development often has costs higher than custom development, goodness of fit lower than custom development and schedules that may extend as long as custom development. MOTS development is often the worst of all worlds.

COTS Customization

But there is an intended range of COTS customization. When do you cross the line from simply customizing your off-the-shelf packaging to rewriting it? Knowing where the line is requires reading the standard manual of the standard version and applying only the customization described there and used by all customers. This can be tricky as documentation may be imperfect. An expert may be required to make this judgement. Alternately you can review your maintenance costs versus the average customer, and note if your costs are very high.

Integration Technology

A wide range of integration technology exists to take COTS products, and some custom stuff, and to interconnect these via published standard production interfaces. This includes a range of EAI, ESB, workflow BPM and middleware packages. All these are intended to work with existing interface points intended by the vendor. Using these as intended does not lead to the MOTS dilemma.

SOA

Many have been fooled into MOTS in the rush to implement SOA, they modify COTS to meet the needs of a predetermined set of web services. Such services are often identified by governance

processes carried on in isolation, academic exercises, that are forced on existing legacy applications. This may not meet the business objectives that drove you toward SOA. The result bypasses provided integration points and modifies the COTS to do so.

Moral

If you want the low cost of COTS software, you might consider not modifying it too much.

3.3 THE ALBINO RHINO HUNT, JUNE 13, 2015



There is an economic contraction in government contracting, driven by government austerity. While government should be spending in truckloads to address real problems like cyber-security, border security, reallocation of global forces, industrial policy (to improve our workforce and industry versus global competition) and other factors, it is instead focused on competing with industry and internalizing jobs. Economic signals to industry are thereby confused and muddled by the current politics.

As a result, the vast marketing and problem-solving machine that serves the US Government, who put men on the moon and won the Cold War, is now focused on chasing a mythical beast. This [chimera](#) is constructed of wispy rumors of opportunity, rarely seen in actuality. It is the Big Agile Cloud Data system, or as architect [Kevin Lorenz](#) puts it: the [Albino Rhino](#).

You can look all day for real appearances of this beast, places where there are real opportunities to build the next huge, cloud based, big-data system using Agile methods. You will not find those opportunities, those jobs, except for sales support chasing that [mythical beast](#). Career professionals with a long-term perspective are sometimes reticent to join this [wild hunt](#). Clearly the hype driven quest is temporary to some extent. It is not sustainable.

The ironies in this mythology are significant:

- Efforts to build really big systems using Agile methods tend to fail often;
- Precious data would be safer away from the public cloud;
- Effective methods to manage or analyze "Big Data" are not Agile in nature.

But this is the big hope, the only big game in town. So the next time you see that marketing-sales professional speaking of all of this in combination, with a strange fierce but hungry gleam in the

eye, do not confront him or her. Do not suggest that their quest is myth. They might become violent. Simply find an excuse to back away quietly.

3.4 IS THE CLOUD NEW, JUNE 29, 2015

Cloud vs Mainframe

Technology	SaaS or Equivalent	PaaS or Equivalent	IaaS or Equivalent
2015 Cloud	SAS, SAP, email, databases, various editors	VMware, Azure	Cloudability, Cloudfy, Enstratus, Chef, Puppet
1980's Mainframe	SAS, SAP, Sendmail, Informatica, DB2 and other databases, various editors...	VM, VM/370 and equivalents	See JCL, JES2 and equivalents

How does the mainframe environment of the 1980s compare to the current cloud environment? I will present a brief comparison based on current cloud terminology and offerings. This post will be based somewhat on my opinions, as comparison and evaluation will be required.

SAAS

In Software as a Service you buy the application and any supporting computing resources. You are a user of the application and manage nothing, unless you arrange to perform some application level administration. This may include email packages, document editing, statistical analysis, database applications like ERP, or almost any other software application. Such customer usage of the applications run in the data center were the "bread and butter" of the mainframe era.

PAAS

In Platform as a Service you rent a virtual machine. You own and administer that virtual machine. In the mainframe era the virtual machine environment of IBM was called VM, and you could run any IBM operating system (and later UNIX) on top of it. You could "peg up" a virtual machine and leave it running as long as you like, or demand one "on the fly". Today we buy VMware or an equivalent to do similar things.

IAAS

Infrastructure as a Service allows you to request machine resources rapidly and then have them provisioned for you "on the fly". In the mainframe era, a sophisticated mechanism called JCL (Job Control Language) automated this. A sophisticated version was JES2 (Job Execution System 2), which provided for all the cloud related benefits like resilience, broad network access, resource

pooling, rapid elasticity and measured service with billing. Mechanisms to do the same in today's cloud are slightly less flexible and sophisticated IMO. These include Chef and Puppet.

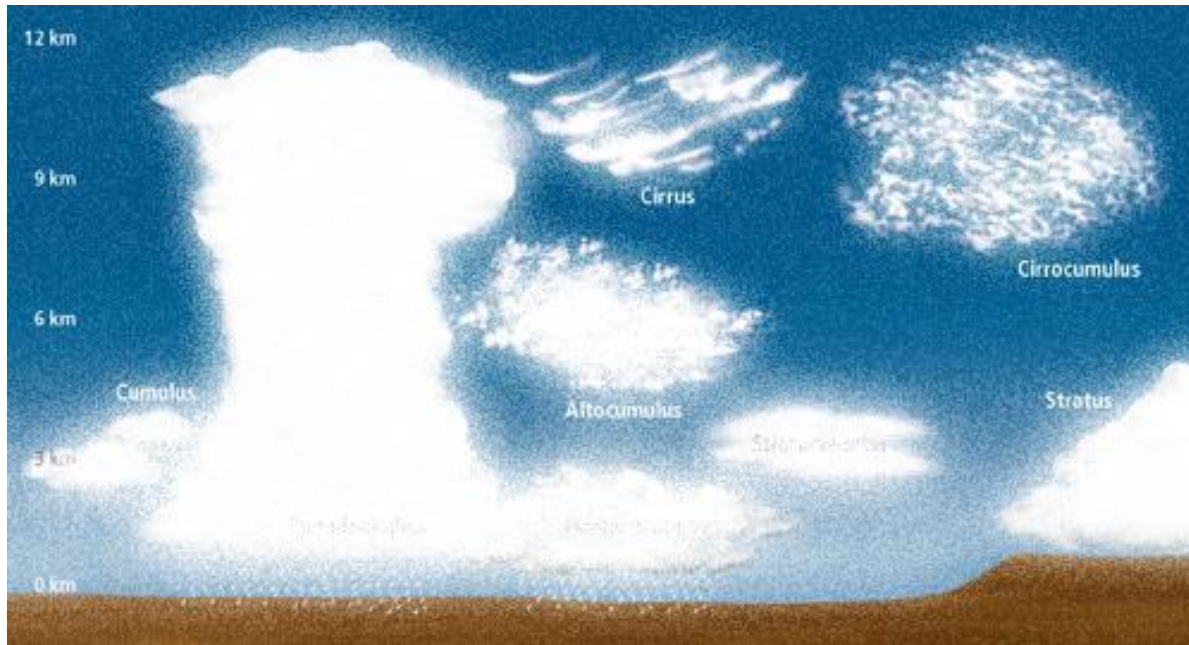
WHAT HAS CHANGED

Of course the operating systems and applications have changed. The Internet has replaced SNA. Process isolation and built-in security in the environment have decreased IMO. Reliability and availability have mostly decreased.

I have to conclude that no part of the cloud concept is particularly new, although the products are.

"The more things change the more they are the same." Alphonse Karr

3.5 CLOUD SERVICE CATEGORIES, AUGUST 23, 201



US Government Image from NOAA.

Cloud terminology is mysterious to some. Here are my own categories for cloud services. My hope is that this might help you think about this set of services.

TYPE I

You buy a virtual machine hosted by someone else. You are responsible for that machine, its maintenance and configuration. This usually comes with some supporting network and storage, maybe load balancing. (This is also called IaaS or Infrastructure as a Service.)

TYPE II

You buy only "cloud storage" and maybe email. This is very common right now, often free for limited use.

TYPE III

You buy a Web Server that comes with the operating system, a development environment, e-mail, a database, maybe e-commerce support and web statistics and such. Usually they will supply upgrades and patches. (This is also called PaaS or Platform as a Service).

TYPE IV

You buy a COTS application or suite (such as ERP) hosted on their virtual machine. (This is often called SaaS or Software as a Service.)

TYPE V

You buy IP telephony, videoconferencing, instant messaging, email, and other human to human communications. Sometimes this is called Unified Communications as a Service. In the future this might include machine to machine communications, ESB/EAI or Integration as a Service (Type IVb, aka Integration as a Service).

TYPE VI

You buy whatever virtual servers you need from a customer menu application, with a credit card (or account), and it shows up almost immediately. They maintain and patch whatever they supply. This might be various platforms, web servers, databases, firewalls, email, SharePoint, unified communications, integration platforms, whatever. Different software may be available on different operating systems. This is the future vision, and is sometimes associated with names like "multicloud", "intercloud", "distributed cloud", "automated user provisioning" and other buzzwords.

COMMON ELEMENTS

All of these run on virtual servers, hosted out there by someone else, in possibly more than one location that may move. All presumably will add resources and capacity as needed, transparently, and bill you for what you use.

All promise near infinite safety, which is ridiculous as the Internet between the cloud offering and your office location is usually not that safe, even with a VPN (Virtual Private Network). Some provide only standardized security controls, but you may need to add more application specific controls for hosted applications especially.

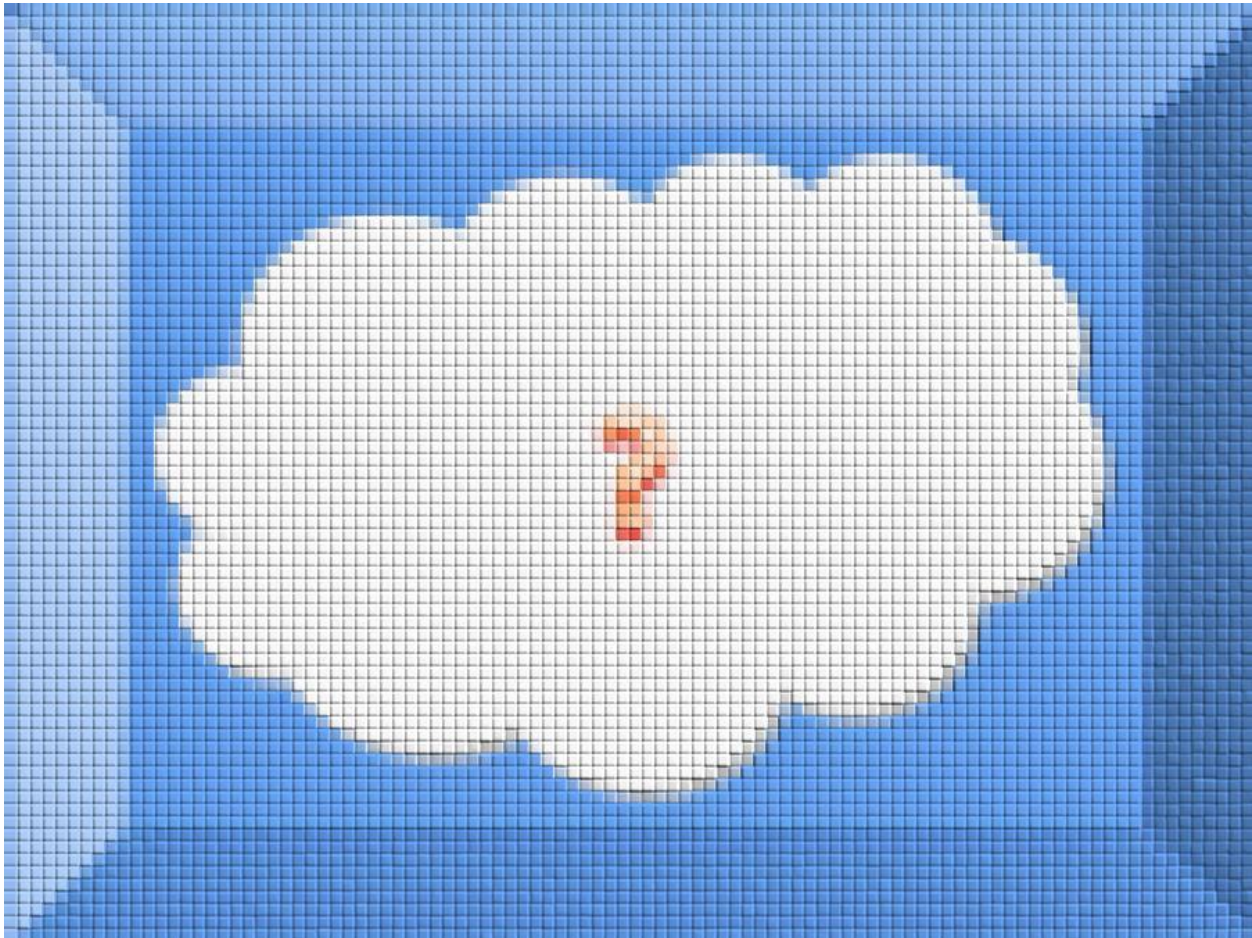
Any may be either public cloud, or private cloud. You might buy different pieces from different vendors (hybrid cloud), and there are initiatives to make parts of that inter-operable (intercloud).

CLOUD BENEFITS

The cloud is sold as greatly reducing costs and effort. The underlying virtualization technology has the capability to do this. First many initial organizations must move to the cloud. Then economy of scale occurs, and prices drop. After that, a shakeout occurs, as we do not need thousands of cloud providers. Then, if enough competition remains, prices will drop again through further economies of scale. At that point we will have realized the scale of price drops predicted.

The rush to the cloud is slower than predicted. Security concerns are part of the delay. Today type I, II, III and V cloud services offer limited promise to greatly drop operational costs. Type IV offers significant potential savings if you do not try to redesign the COTS software for each customer. Type VI remains a future hope.

3.6 THE CLOUD PARADOX, SEPTEMBER 28, 2014



The cloud will save us all. In the cloud we trust. But let me ask a question...

Physical servers are becoming smaller, less expensive. Servers the size of credit cards, complete with processor, IO, memory, hard disk and the whole deal, are available for purchase. Physical server prices are dropping. Physical server density in data-centers is about to skyrocket, limited only by cooling and power management. Soon 10,000 servers in a standard 6 foot tall 19" rack may be commonplace. That is enough to run a major corporation.

Virtualization makes servers cheap by putting multiple virtual servers on a physical server. Prices and effective sizes of servers then drop significantly again.

The cloud is the idea that we will not know where our virtual server is at any moment, it might be on any physical server. It is also the idea that all these servers will be housed in some big data center.

PERHAPS NOT SO BIG

Physical servers may decrease in size 2 orders of magnitude, and virtualization may effectively drop the size of servers needed another order of magnitude. Instead of a warehouse full of servers, you

may need only a garage, Google's square miles of servers may drop to the size of a small department store.

Here is my question: At those densities who needs to centralize? Every big organization can clear a few offices and install its own cloud.

Moreover, if servers are as cheap as a wireless router, where are the cost savings for centralization? The economies of scale do not seem to apply to the hardware once it becomes a commodity as cheap as dirt.










Perhaps we are really talking about saving labor to administer and maintain those servers in the cloud? Well you could buy remote administration and maintenance from a vendor for your in-house little cloud farm just as cheaply. The same economies of scale for labor would apply to such a service without involving the hardware at all.

There are some important security advantages to having physical possession of your own servers and data. ACME may not want to part with that control, if the cost savings are not greatly in favor of further centralization.

I have seen the emperor, and I am not sure he is clothed.

3.7 SECURITY VS CLOUD, AUGUST 23, 2014

CLOUD TECHNOLOGY SELECTOR

PRACTICE SAFE - DATA	PUBLIC CLOUD	PRIVATE CLOUD	AVOID CLOUD
HIGH CONFIDENTIALITY			
MEDIUM CONFIDENTIALITY			
LOW CONFIDENTIALITY			

When should you trust your data to the cloud? That depends on what you mean by "cloud". Below is a rule-of-thumb the pros sometimes use.

If you have not noticed, just about every security technology we use on the internet has been compromised, has unaddressed vulnerabilities, has back doors, you name it. The encryption that secures your data mostly doesn't. Root certificates have been compromised. Even DHS has lost user data. Password data has been spilled in floods. The permissions repository you trust is open to access. It is all shot through with vulnerability. So here is the simple rule.

If the data you will be placing in the cloud has low confidentiality requirements, if access by the enemy is not a big problem, then you can put that data on shared public cloud servers. If it is "personally identifying" or purchase data, having medium confidentiality requirements, you can put it on the private cloud. If it is banking bulk financial data or national security data, you should not put it on the cloud at all.

SOME CLARIFYING DEFINITIONS

- **Low Confidentiality Data:** Data where the release or leakage is not that important. Nobody dies. Nobody loses much money.

- **Medium Confidentiality Data:** Data of a protected category, such as volume PIV, financial or credit card data. Someone might be sued. Someone might go to jail.
- **High Confidentiality Data:** If this data gets out, people are injured or die, perhaps in significant numbers. Countries or companies may fall, destroyed. War could result.
- **Public Cloud:** An offering of a commodity provider in which your data is stored on the same servers, with the same security controls as all other customers.
- **Private Cloud:** An offering in which you get physically separated servers and network segments as well as custom security controls for your application (aka an "enclave"). This may be in your organizational data-center or a 3rd party data center. The Amazon private cloud offering can be viewed here: <http://aws.amazon.com/vpc/>
 - Note the offering allows you to use servers dedicated to you as a customer.
- **Not On the Cloud:** Servers you own and control not connected to the Internet. In some extreme cases, not connected to any network. All personnel with physical or logical access to your data (including all administrative or maintenance access) are reviewed and approved by your security policies and processes. All access to your data, physical or logical, is controlled and monitored by you. (October 15, 2014... DoD is now considering servers that they do not own, but do control in controlled secure space, for this application. These servers would still not be connected to the general Internet, but on a TS secure intranet.)

3.8 ESSENTIAL CLOUD ARCHITECTURE FOR CUSTOMERS, MARCH 21, 2015

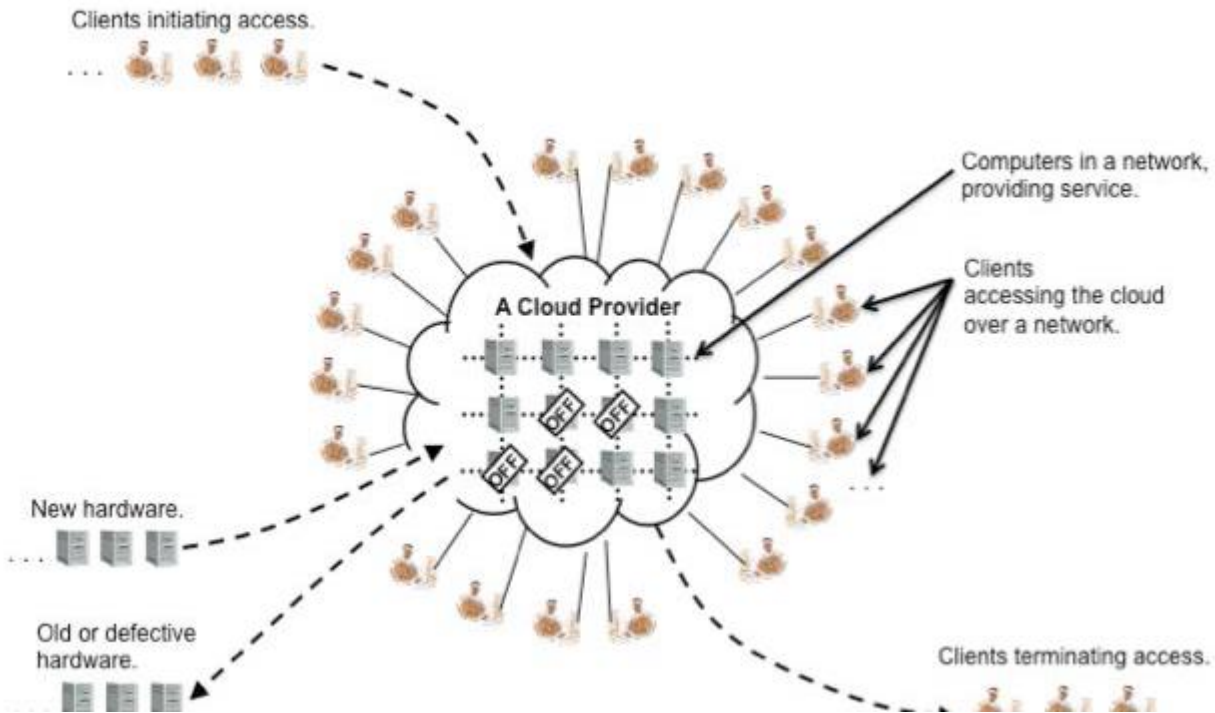


Image from US Government, in NIST documents cited below.

Have you seen the NIST documents clarifying Cloud Computing? These things are great. They specify what the terminology is, and even have diagrams.

- <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
- <http://csrc.nist.gov/publications/nistpubs/800-146/sp800-146.pdf>

There are very clear definitions:

PUBLIC CLOUD:

Most articles and advertising promote the public cloud. This is some vendor offering.

NIST DEFINITION: "Public cloud. The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider."

PRIVATE CLOUD:

By contrast the most secure option here is the private cloud.

Private cloud. The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

It is important here to remember that while either on premises or off-premises, construction may still be a private cloud; and, while either you own and control it, or a vendor does, it will still be a private cloud, these decisions have extreme impact on security. If the hardware itself is accessible to parties not cleared by you, you may assume they can gain access to your data. They can take it, copy it, crack it, and use it with no difficulty to some difficulty because they can touch your platforms with their very real hands.

The most secure private cloud will be on your site, using only people you have cleared.

BETWEEN THESE:

There are two more options described. These each require some detailed security engineering for your configuration. Such engineering should be mostly provided by your cloud vendor, it's their service - right?

COMMUNITY CLOUD. THE CLOUD INFRASTRUCTURE IS PROVISIONED FOR EXCLUSIVE USE BY A SPECIFIC COMMUNITY OF CONSUMERS FROM ORGANIZATIONS THAT HAVE SHARED CONCERNS (E.G., MISSION, SECURITY REQUIREMENTS, POLICY, AND COMPLIANCE CONSIDERATIONS). IT MAY BE OWNED, MANAGED, AND OPERATED BY ONE OR MORE OF THE ORGANIZATIONS IN THE COMMUNITY, A THIRD PARTY, OR SOME COMBINATION OF THEM, AND IT MAY EXIST ON OR OFF PREMISES.

HYBRID CLOUD. THE CLOUD INFRASTRUCTURE IS A COMPOSITION OF TWO OR MORE DISTINCT CLOUD INFRASTRUCTURES (PRIVATE, COMMUNITY, OR PUBLIC) THAT REMAIN UNIQUE ENTITIES, BUT ARE BOUND TOGETHER BY STANDARDIZED OR PROPRIETARY TECHNOLOGY THAT ENABLES DATA AND APPLICATION PORTABILITY (E.G., CLOUD BURSTING FOR LOAD BALANCING BETWEEN CLOUDS).

FEDRAMP

The FedRAMP program provides cloud application security certification. It has GSA backing. If you buy a COTS vendor package that has been FedRAMP certified, then a wide range of security controls have been implemented.

If you buy or build a custom software package and run it on FedRAMP platform, then the platform and not the application have been certified for a range of security controls.

In either case, some controls dependent on your data and organizational processes are NOT COVERED BY FedRAMP. For example you may have unique separation of duties rules, or access role restrictions, not built in to COTS and not known by GSA or the FedRAMP vendor. They could not have been implemented as you need them, until and unless you are involved in the process (during or after acquisition).

Back to NIST:

In SaaS you have that opportunity to buy a COTS application. Some security controls may be built into that application:

SOFTWARE AS A SERVICE (SAAS). THE CAPABILITY PROVIDED TO THE CONSUMER IS TO USE THE PROVIDER'S APPLICATIONS RUNNING ON A CLOUD INFRASTRUCTURE. THE APPLICATIONS ARE ACCESSIBLE FROM VARIOUS CLIENT DEVICES THROUGH EITHER A THIN CLIENT INTERFACE, SUCH AS A WEB BROWSER (E.G., WEB-BASED EMAIL), OR A PROGRAM INTERFACE. THE CONSUMER DOES NOT MANAGE OR CONTROL THE UNDERLYING CLOUD INFRASTRUCTURE INCLUDING NETWORK, SERVERS, OPERATING SYSTEMS, STORAGE, OR EVEN INDIVIDUAL APPLICATION CAPABILITIES, WITH THE POSSIBLE EXCEPTION OF LIMITED USER SPECIFIC APPLICATION CONFIGURATION SETTINGS.

In PaaS you have the opportunity to deploy whatever application you wish to buy or build, which may come with or without some security engineering built in:

PLATFORM AS A SERVICE (PAAS). THE CAPABILITY PROVIDED TO THE CONSUMER IS TO DEPLOY ONTO THE CLOUD INFRASTRUCTURE CONSUMER-CREATED OR ACQUIRED APPLICATIONS CREATED USING PROGRAMMING LANGUAGES, LIBRARIES, SERVICES, AND TOOLS SUPPORTED BY THE PROVIDER. THE CONSUMER DOES NOT MANAGE OR CONTROL THE UNDERLYING CLOUD INFRASTRUCTURE INCLUDING NETWORK, SERVERS, OPERATING SYSTEMS, OR STORAGE, BUT HAS CONTROL OVER THE DEPLOYED APPLICATIONS AND POSSIBLY CONFIGURATION SETTINGS FOR THE APPLICATION-HOSTING ENVIRONMENT.

In IaaS you also deploy the operating system, and perhaps software routing, firewalls and whatever. Security is more in your hands than in that of the cloud vendor.

“INFRASTRUCTURE AS A SERVICE (IAAS). THE CAPABILITY PROVIDED TO THE CONSUMER IS TO PROVISION PROCESSING, STORAGE, NETWORKS, AND OTHER FUNDAMENTAL COMPUTING RESOURCES WHERE THE CONSUMER IS ABLE TO DEPLOY AND RUN ARBITRARY

SOFTWARE, WHICH CAN INCLUDE OPERATING SYSTEMS AND APPLICATIONS. THE CONSUMER DOES NOT MANAGE OR CONTROL THE UNDERLYING CLOUD INFRASTRUCTURE BUT HAS CONTROL OVER OPERATING SYSTEMS, STORAGE, AND DEPLOYED APPLICATIONS; AND POSSIBLY LIMITED CONTROL OF SELECT NETWORKING COMPONENTS (E.G., HOST FIREWALLS).”

Essential Characteristics

NIST defines a set of essential characteristics of the cloud designed to save you money. However, with PC servers moving toward a scale of thousands per rack, and prices headed to micro-cents per gigaflop-second, these money saving mechanisms may become unimportant - as may also parts of the cloud infrastructure.

ON-DEMAND SELF-SERVICE. A CONSUMER CAN UNILATERALLY PROVISION COMPUTING CAPABILITIES, SUCH AS SERVER TIME AND NETWORK STORAGE, AS NEEDED AUTOMATICALLY WITHOUT REQUIRING HUMAN INTERACTION WITH EACH SERVICE PROVIDER.

BROAD NETWORK ACCESS. CAPABILITIES ARE AVAILABLE OVER THE NETWORK AND ACCESSED THROUGH STANDARD MECHANISMS THAT PROMOTE USE BY HETEROGENEOUS THIN OR THICK CLIENT PLATFORMS (E.G., MOBILE PHONES, TABLETS, LAPTOPS, AND WORKSTATIONS).

RESOURCE POOLING. THE PROVIDER'S COMPUTING RESOURCES ARE POOLED TO SERVE MULTIPLE CONSUMERS USING A MULTI-TENANT MODEL, WITH DIFFERENT PHYSICAL AND VIRTUAL RESOURCES DYNAMICALLY ASSIGNED AND REASSIGNED ACCORDING TO CONSUMER DEMAND. THERE IS A SENSE OF LOCATION INDEPENDENCE IN THAT THE CUSTOMER GENERALLY HAS NO CONTROL OR KNOWLEDGE OVER THE EXACT LOCATION OF THE PROVIDED RESOURCES BUT MAY BE ABLE TO SPECIFY LOCATION AT A HIGHER LEVEL OF ABSTRACTION (E.G., COUNTRY, STATE, OR DATACENTER). EXAMPLES OF RESOURCES INCLUDE STORAGE, PROCESSING, MEMORY, AND NETWORK BANDWIDTH.

RAPID ELASTICITY. CAPABILITIES CAN BE ELASTICALLY PROVISIONED AND RELEASED, IN SOME CASES AUTOMATICALLY, TO SCALE RAPIDLY OUTWARD AND INWARD COMMENSURATE WITH DEMAND. TO THE CONSUMER, THE CAPABILITIES AVAILABLE FOR PROVISIONING OFTEN APPEAR TO BE UNLIMITED AND CAN BE APPROPRIATED IN ANY QUANTITY AT ANY TIME.

MEASURED SERVICE. CLOUD SYSTEMS AUTOMATICALLY CONTROL AND OPTIMIZE RESOURCE USE BY LEVERAGING A METERING CAPABILITY¹ AT SOME LEVEL OF ABSTRACTION APPROPRIATE TO THE TYPE OF SERVICE (E.G., STORAGE, PROCESSING, BANDWIDTH, AND ACTIVE USER ACCOUNTS). RESOURCE USAGE CAN BE MONITORED, CONTROLLED, AND REPORT.

Server Miniaturization

Every technology has its lifespan. The benefits of the cloud diminish as servers are miniaturized, commoditized and made inexpensive on grand scales. If servers are as cheap and ubiquitous as popcorn kernels there is less need to manage and share server capacity as a valuable commodity. The big barriers to this future are cooling the server cards and loading them into server racks by the hundreds (packaging). You can keep the virtualization on some of them. If you want to leapfrog the cloud, look at very small, high density server farms. Don't believe me? Look here.

- http://www.deployabletechnologies.com/BioDigital_PC.html
- <http://www.pcgamer.com/credit-card-sized-odriod-c1-is-a-quad-core-computer-for-only-35/>
- <http://www.engadget.com/2012/02/29/raspberry-pi-credit-card-sized-linux-pcs-are-on-sale-now-25-mo/>
- <http://liliputing.com/2014/07/solidruns-hummingboard-credit-card-sized-computer-launches-for-45-and-up.html>

VENDOR CLOUD CERTIFICATIONS

Don't bother with most vendor cloud certifications unless you intend to own your very own cloud. These certifications allow you to set up and administer the servers in the cloud, exactly what you are outsourcing. Only a few cloud vendors will need these folks. It's mainly administration, not architecture. Customers like you should be isolated from all that, that is the story they are selling - right?

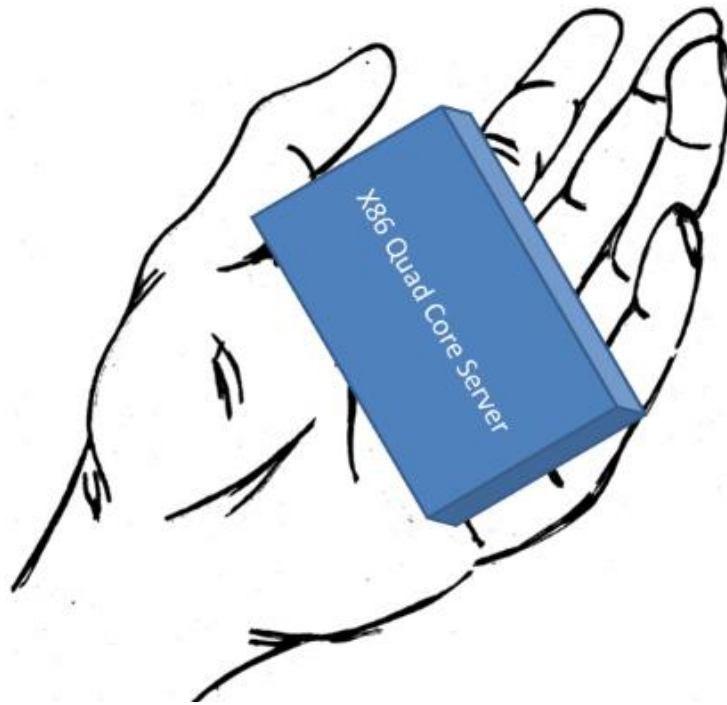
ARCHITECTURE

As for customer-side architecture, nothing much changes. You still have the network on premises, COTS apps, custom apps, business processes, and the only thing that really changes is that you have virtual servers. Desktops and tablets and mobile devices and sensors and peripherals are not in the cloud, right? (You might also get involved in the border protection securities isolating your cloud a bit. Mostly you outsourced that and should have a clear SLA).

THE EMPEROR HAS NO CLOTHES

The good news is that unless you are a cloud vendor, or you are managing the internal on-premises physical cloud servers, there is no "cloud architecture" to perform! It's not Cloud specific! It's mostly hype! With a short review of new terminology and procurement related options, your existing architects are good to go.

3.9 THE CLOUD KILLER, JUNE 30, 2015



Imagine an [x86 quad-core server complete with VMware the size of a credit card](#). It exists, and you can buy one today. Go ahead, run 20 virtual servers on it. How about a [quad ARM running the Ubuntu Linux OS](#) for only \$35. How about [less computer for only \\$9?](#) Need more power per card? This one has [192 parallel processor cores for \\$192](#). If these boards catch on, [they will become cheaper yet](#).

The cloud (as an outsourced big ticket item) is based on the idea that it is good to share expensive computer hardware, using parts of physical servers allocated to your use as virtual servers. The whole idea becomes less interesting if servers are the size of credit cards and less than \$100 each, and also support virtual servers. Either way outsourcing administration and support is the same, and available everywhere.

A whole data center big enough to run a very large company could fit in your 2 car garage. The entire Pentagon's data center needs might fit in a basketball court sized IT room, probably less. The hard parts are power and cooling.

The government may have moved too slowly. The whole outsourced cloud thing, and all those supposed cost savings, may already be OBE, Overtaken By Events. Just let a contract for some beltway bandit to install all the servers you're agency may ever need, in miniature format, in some spare office space, at less cost than the cloud vendors can provide that capacity for.

OOPS.

HOW CLOUDS SAVE YOU MONEY

There are several types of "cloud". All save you money in the same way. The infrastructure of a computer data center is expensive, and you share that expense with others. This is possible because the machines there have more capacity than you need,

In fact your capacity needs vary, and sometimes you need a great deal while at others you need little. If you owned all the machines, you would need enough to support your peak capacity. However when sharing, another company can use the capacity you are not using. Their peaks and valleys of need differ from yours. The total number of machines between you is less, thus saving money.

The savings are incrementally big for the first few customers, but the savings decrease as the number of customers grows. The big peaks and valleys even out and become flat.

LABOR

Labor is often mistaken as a major cost saving for the cloud, but this is only true for some offerings. In private clouds you can own the machine and outsource labor, and this is true if you own the machines or if a cloud vendor owns the machines. It is "a wash", a constant in the cost comparison.

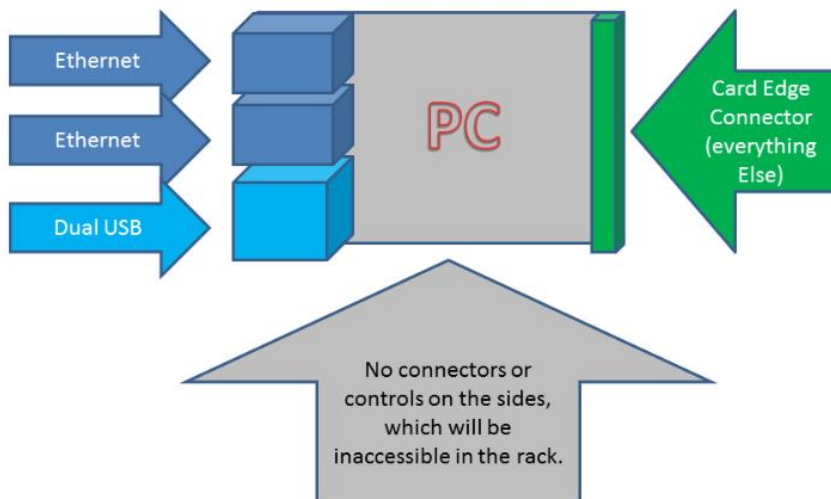
SPECIFIC IMPACT

There is a type of cloud called a "private cloud" where you reserve a set of computers for your own organization. Cloud vendors seek to provide these computers for you. However the argument for this decreases if your computer hardware is very small and fits in your own facilities easily.

In fact recent DHS testimony before congress described creating secure enclaves. These would be private cloud data centers. The most secure type would provide access to only cleared personnel,

cleared by the customer, inside secure facilities controlled by the customer.

Approximate Rack Card Configuration



Therefore the main impact should be in bringing the majority of private clouds in-house, to large customers. However, even the public cloud concept may be endangered by such cheap, ubiquitous servers.

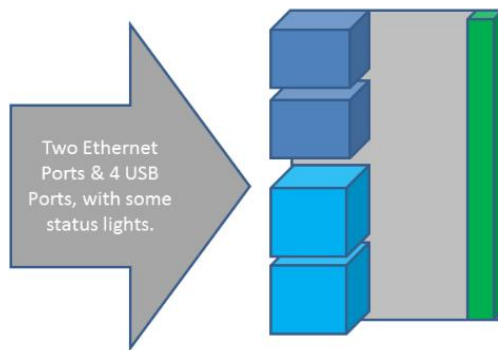
REMAINING BARRIERS

There are four remaining engineering barriers before you will see data centers made of miniature PCs everywhere. They are surprisingly simple in nature.

The first is the card form factor: most of the existing cards are configured as toys. A professional offering for mass rack mounting would place one or two network connections opposite the backplane for connection to the patch panel (network connections would replace any more general use bus connections). Two to four USB connections would also be available opposite the backplane, allowing for server administration operations. A few status lights on that side would help (as would built in self-test). The next barrier is producing a backplane to connect many such miniature

servers in a rack. Up to 2000 physical servers might fit in one standard 6' x 19" rack.

Better Yet, Turn it Sideways



The next two difficulties are slightly more formidable. Cooling such a collection of servers is a problem. However, cooling mechanisms from past mainframe technologies can be applied. Lastly, these things require custom power supplies in the rack, with redundancy. So many servers means a good deal of power will be required. One reasonably sized R&D budget would clear all four of these issues up.

3.10 WHAT IS BIG DATA & WHY NO-SQL?, AUGUST 23, 2014



The marketing term "Big Data" is everywhere. Presumably you need a new breed of technology called "No-SQL" to accommodate this "Big Data". This is what the marketing implies. Let us examine this briefly.

The existing main applications for data management today are called RDBMS (Relational Database Management Systems). Presumably, as these are old hat, they have some limit regarding the total amount of data they can manage. What is this limit? How much data must you have to require this new technology?

- The top RDBMS offerings have the following limits in total data:
- Oracle: Unlimited (with 4GB blocks per tablespace);
- MS SQL Server: 524,272 TB (32 767 files * 16 TB max file size);
- Informix: about 128 Petabytes
- Ingress: Unlimited
- MySQL 5: Unlimited
- Postgress SQL: Unlimited

I admittedly did not spend too long on this, and I just used Wikipedia figures. You can check it there. With 7 billion people on the earth, poor MS SQL Server could only manage 500 Megabytes of unique information for each man, woman and child on the planet. That is the smallest listed. Most have no limit.

Clearly the issue is not that there is data so big that the existing RDBMS technology cannot manage or store it. This is pure hype. Nonsense.

The issue may be more that at those sizes the use of an RDBMS becomes slow. It may be performance we have a problem with. This is more promising.

For many years we have known that the RDBMS is slow. Most of the delay occurs in the SQL front end. SQL, a very powerful non-procedural language to manage data, requires a long time to convert what you have asked for into an access plan and execute. It takes so long that nearly all RDBMS save old queries that you might issue again, or parts of them, to optimize this process.

To avoid this processing time you can use stored queries of various kinds in most databases. Some other "records managers" used the option to provide an SPI (application programming interface) so that you could bypass SQL for speed. Among these was the famous B-Trieve, nearly a legend in its time going back decades. Today it is part of pervasive SQL and also has a SQL front end, a natural evolution to greater functionality.

IBM offered the IMS database, which was known for speed on large datasets. It was non-relational. It was used on the Apollo moon rocket launches, beginning in 1966.

The idea of speeding up data access by avoiding SQL compilation is not new. Use of non-relational database technologies is also decades old.

Hmmm, maybe the new XML paradigm has spawned new approaches to data management that are without precedent, or more efficient?

David Childs formerly of U-MICH is a mathematician, mostly retired now. He noted, back in the early days of the database, that SQL was based on a subset of possible set operations in set theory. What would happen is we used all the available set functions, he asked?

He wrote many papers on massive data-sets in the 1970s and through to recently. He constructed an extended set theory (XST) many years ago, describing how data access could be improved by use of improved mathematics. Following him several developers produced tool-sets to manipulate datasets using XST. When XML came out, this was a natural combination, and development redoubled. The results were lightning fast. While not really transactional, these tool-sets remain unequaled for analytic applications (OLAP).

Tools for manipulating XML to more rapidly access data have a decades old history. However XML tools can provide speed advantages in selected applications.

What then are these selected applications for which these no-SQL technologies may be used to advantage?

Analytic applications, like data warehouses or data-marts are perfect for taking advantage of XST and other less general (garage development) tools to query XML datasets. Using an RDBMS or other means to normalize the data before transferring it to the analytic tools is often a good idea, or data anomalies and quality issues may occur.

When you have a high speed input feed, sometimes you use a database as a staging buffer. Data is later input and normalized and indexed by a slower process into an RDBMS. This is a perfect application for Mongo DB, for example.

One truly new aspect of these new technologies is parallelism. Hadoop can run on many distributed servers. You can run distributed queries. However this is no longer NO-SQL per-se, as Hadoop has adopted a SQL front end to provide the greater expressive power of SQL. This is much like oracle grid technology in some ways.

<http://arstechnica.com/information-technology/2013/07/the-hot-new-technology-in-big-data-is-decades-old-sql/>

So this new technology is really good in niche applications. What about replacing the RDBMS in ordinary, forms based transactional applications? Well, there is perhaps a big cost-advantage as some of these no-SQL products are open source. But there is an important caveat...

Many years ago, mathematicians and theoreticians and computer scientists worked out what you must have for a database to perform correctly in transactional, forms-based applications or similar. The result was described in an acronym called ACID. (see my previous post on same.) Some NO-SQL database products are not ACID compliant, and will produce incorrect results in some applications. Producing correct results takes undue care without ACID.

Those NO-SQL tools that are ACID compliant seem to be adopting SQL just as Hadoop did, becoming just another RDBMS with some new features.

FOR MAINSTREAM TRANSACTIONAL APPLICATIONS, OLTP, NO-SQL TOOLS ARE MERGING INTO THE RDBMS MARKET, OFFERING RDBMS ACID FUNCTIONALITY AND THE RDBMS REMAINS KING.

This is just my viewpoint folks, let me know if I missed something. However, it seems the NO-SQL Big Data hype wave will pass, leaving behind new RDBMS offerings and some niche application tools.

In the end, these "Big Data" applications are just more of what came before, and the requirement for brand new technology is somewhat limited.

So again I ask, what is Big Data, and why NO-SQL?

Big data is not about technology, it seems. There are few limits to transcend, no real barrier. It is about creating huge databases tracking things ignored before. It seems to be finding these unexplored new datasets, and convincing people to track them. Marketing.

Go ahead, market people you vendors! Everybody needs to make a living! Just quit telling me it's a revolution.

Maybe I didn't call the Emperor first:

http://www.slate.com/articles/technology/bitwise/2014/07/facebook_okcupid_user_experiments_ethics_aside_they_show_us_the_limitations.html

...And then the ethics problem

- <http://www.amazon.com/Ethics-Big-Data-Balancing-Innovation/dp/1449311792>
- <http://radar.oreilly.com/2012/06/ethics-big-data-business-decisions.html>
- http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2384174
- <http://www.forbes.com/sites/oreillymedia/2014/03/28/whats-up-with-big-data-ethics/>
- <http://www.datasociety.net/initiatives/council-for-big-data-ethics-and-society/>
- <http://www.technologyreview.com/news/424104/what-big-data-needs-a-code-of-ethical-practices/>
- <http://www.tnwinc.com/2737/privacy-and-data-information-security-training/>
- http://www.cmo.com/articles/2014/4/3/big_data_ethics_tran.html
- <http://searchcio.techtarget.com/opinion/Cool-or-creepy-The-ethics-of-big-data-is-on-the-table>

The Google query "big data ethics" returns: "About 40,500,000 results (0.27 seconds)"

Maybe, just maybe, people avoided many of these big databases in the past because those applications are not ethical. Maybe they think the world has changed. Maybe it hasn't. I suppose we can watch Facebook and see what happens.

http://www.washingtonpost.com/opinions/facebooks-study-crosses-an-ethical-line/2014/07/02/61ff3502-0130-11e4-8572-4b1b969b6322_story.html

3.11 BASIC TRUTH: OLTP AND OLAP, JUNE 1, 2014



Big Data or not, this has not changed. Conventionally there are two major categories of database use.

ON-LINE TRANSACTION PROCESSING (OLTP)

This category involves application front ends and forms. Each user reads records in forms, creates new records, updates records and deletes records, may lock cursors, and expects transactional integrity. The term CRUD (Create, Read, Update, and Delete) refers to the four basic operations in OLTP. The term ACID refers to the four fundamental properties of a reliable database:

- Atomicity: Each transaction is complete or does not occur, no partial completion.
- Consistency: Each transaction takes the database from one consistent state to another.
- Isolation: If concurrent transactions occur, it is as if they occur in sequence, no differences in results are introduced.
- Durability: Once a transaction is completed, it does not disappear with a reboot or if the power fails.

ON-LINE ANALYTIC PROCESSING (OLAP)

OLAP is different. In OLAP activities are not typical small, short user transactions. Transactions may lock multiple tables for minutes or even hours as complex joins and comparisons occur. This

supports reports, dashboards, analytic tools, statistical tools, visualization and other applications that analyze data in detail.

SEPARATING OLTP AND OLAP

For non-trivial applications, one database may be split into two for the OLTP and OLAP side of the application. In an enterprise, many OLTP databases and several OLAP databases may exist without a 1 to 1 correspondence. This kind of separation dramatically improves your performance and price to performance, eases your administration and makes development far easier.

ETL

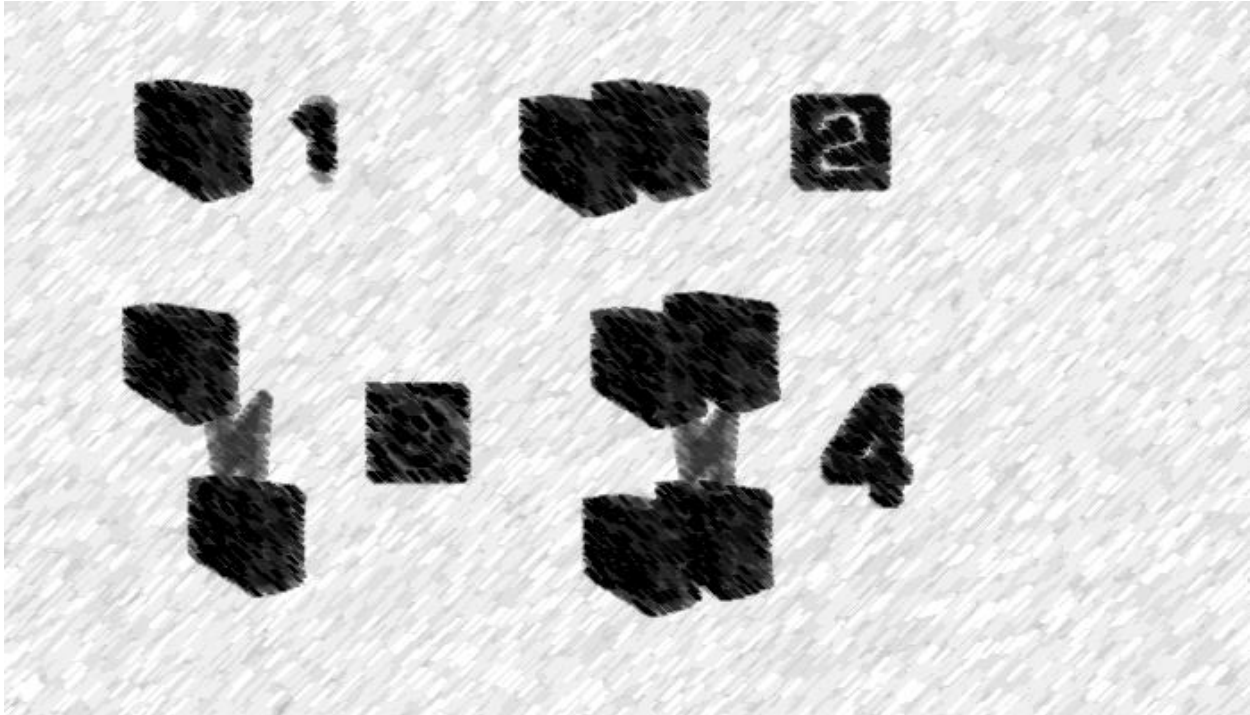
Extract, Transform, Load tools are available to transfer your OLTP databases to your OLAP databases. They are inexpensive, and often come free with your database as a supporting tool. Almost any transformation can be performed in an ETL tool. These typically move data overnight in a bulk transfer. This can be thousands of times faster and more efficient than transfer via multiple queries as each database can be put in a special mode where other users are ignored during the bulk transfer.

TIPS

- 1) Non-RDBMS databases without ACID transaction support should not be applied to OLTP application development. This may include various cool new database technology. These are often intended for OLAP use only.
- 2) It is possible to use web services to move data from your OLTP databases to your OLAP databases. This is generally an insanely stupid idea for the following reasons: First these services often do not bother to lock the databases in single user mode creating massive inefficiency. If you are writing web services to replace ETL you might want distinct services that lock the database. Second, ETL tools are very inexpensive and sometimes free, and it can do almost any required function. Why spend extra money programming to accomplish nothing more?
- 3) One possible exception to (2) above is a workflow in which data quality issues are corrected with manual assistance. You may be able to do the same with error tables for suspect records in your database structure and ETL, however.
- 4) If despite the above you need to perform ETL functions via some homegrown composed service you should be aware that the most powerful transformation functions are beyond XSLT and common XML tools. You need to check out Dr. David Childs at University of Michigan and extended set theory or XST. This is technology so far ahead and beyond the norm that it causes many to walk away muttering of voodoo. It is, however, highly credible and real and Dave Childs has been working with it since the days of CODASYL.

This material was first posted on my personal blog site in [Database & BI](#) and tagged [ACID](#), [BI](#), [CRUD](#), [Database](#), [OLAP](#), [OLTP](#), [WTL](#) on [3March2013](#).

3.12 FUN WITH DATABASE SERVERS, SEP 4, 2015



In an enterprise application the database server is a central resource. Enterprise applications are often mission critical. If you need one big data model, how can you keep it working reliably? Let me walk you through an introduction.

SCENARIO 1

One Server, somewhere in your data center (virtualized or not) contains your (still most often relational) database. This is our base scenario. You might add dual power supplies and disk arrays to keep this server running reliably. We will assume UPS and even a site backup generator. Dual redundant LAN connections have also been used, just to be certain. Multiple independent WAN connections (telephone lines) to the site are also often used for reliability.

SCENARIO 2- LOCAL FAILOVER

A second server (or more) can be collocated. The key is the small distance between servers. This allows common distributed database products to operate. When one fails the others keep running, with copies of the database split between them in various product specific ways. You are now protected from a hardware failure of one server. You can also add all the other stuff from scenario 1.

Other technologies allow two physical servers to share a common disk, and when one fails the other starts. Only a few transactions will be lost. They call this a "hot spare". A second virtualized server can be started on failure with backups every so often--- we will cover warm spares below. Using warm or cold failover for local site servers is a half-measure to cut costs.

SCENARIO 3- DISTANT FAILOVER

Local failover is not enough! What if there is a flood there? 'A landslide? 'The site is hit by lightning and a fire starts? What if the Internet fails in that area (this has happened several times in history)? Or maybe terrorizers (George W. Bush term for terrorists).

Because the sites are distant, many common distributed database products will not work when the distance between servers is greater than some maximum. Although bits move through the network at speeds approaching the speed of light, coordinating transactions soon begins to slow the transaction rate of the database.

Products like Hadoop seek to change the equation by providing means to coordinate between databases at arbitrary distance. Some require splitting the database up using some trick applied to the data model. For example all US transactions go in the USA database, all European transactions in the EU database, and updates across them are exceptional cases handled differently. Queries across databases are coordinated more easily.

You can write such code into your application, but most enterprise applications are not written this way from the vendor. A custom application is usually where you will see this technology applied. It is not new technology. The two phase and three phase distributed database commit protocols most often used were developed in the 1980s. Layer 5 of the OSI ISO model of network communications builds in mechanisms for database coordination- though rarely implemented. IBM mainframes had APPC for the same purpose.

If you have a legacy application you may want to consider warm or cold backup. In warm backup incremental updates are shipped to the backup database at intervals, usually electronically. If the interval of backup shipments is 30 minutes, you will lose 30 minutes of transactions and more on failover. If you keep the backups without automatically loading them on the backup server you have cold backup. You may obtain or identify the backup server, or turn it on, only after failure- and then load the data. This can take some time. (For details consult your local CISSP, who is trained in these terms and technologies.)

SCENARIO 4- COMBINE IT ALL

It is best for reliability to use both local distributed databases or hot spares and also distant server failover or database distribution. This is especially true if using warm or cold backup at the distant site. The costs of a local server failure will be contained (dollars, transaction time, restoration time, etc.), and distant failover costs will not be invoked unless needed.

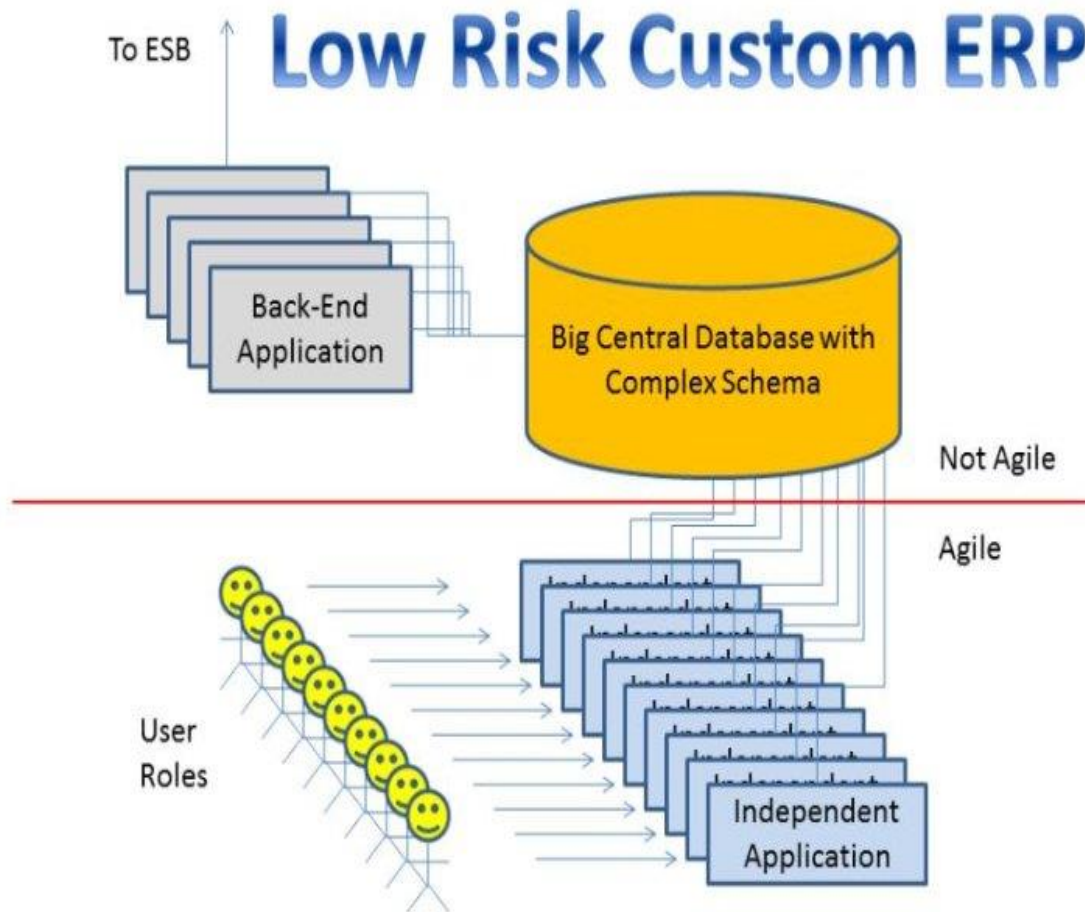
PERSPECTIVE

If your core business relies on these systems, if lives or human suffering may be a consequence of system failure, if these systems are "mission critical", these concepts are important. The "hard" part here is attempting distributed database use across distant sites. It requires skill and study, deliberate design efforts often before coding. Ironically the technologies to produce the greatest organizational agility are not supported by agile development methods, but by more capable methods for handling complexity and size.

There is a Scenario 5, too complex to depict. In this scenario the same large application uses multiple different database technologies for different parts of the data model. These are distributed

differently, or not, as is best for that portion of the application. Real products such as LinkedIn do use this approach. It is hard to manage, and many of the LinkedIn behavior anomalies (bugs) you may experience may be due to this data segregation.

3.13 BUILDING LARGE TRANSACTIONAL APPLICATION SUITES, JULY 15, 2014



Over two decades ago the industry came to a concurrence: It was darned hard to build large, complex application suites like MRP, ERP, CRM and Logistics management. We began avoiding building such applications, at all costs. However building such applications has become easier while we slept. Few have noticed yet. The old notion that these large sets of coordinated applications are as hard as moonshots may be obsolete.

Database Schema

There are now books with CDs in them that contain an enterprise data model for such applications. Your manufacturing model or your inventory model is right in there. You will need to do some de-normalization for efficiency and add a few custom tables to what comes on the CD usually, pretty simple.

Application Development Tools

Microsoft greatly disturbed 2 tier development after VB6, stifling the greatest software reuse scheme ever (OCX), but it is far from dead. You can still get Java Beans and such for your web apps. Two tier development is easy, inexpensive and quick. Some of these are new cloud based development tools.

Simplified Scope

There is a style of large application suite development in which many small teams are assigned to the many roles in the organization. Each team develops a separate independent application focused on one role, one user community. While the overall database model is large, any application only uses a small subset, relevant to the functions performed by the target role. That simplifying assumption puts most of the development into the range suitable for Agile methods. Only the central DBA function and back-end interfaces are non-agile, yet can support scrum. Complexity and risk is greatly reduced. (Incremental builds using Agile or whatever reduce risk even further.)

Three Common, Simplifying Database Features

Most ERP or similar systems will have a central application security schema, an error log schema and a workflow schema. These are used to unify and simplify integration. The last can be had in a book from an IBMer on production workflow. The error log schema is trivial. The security model schemas are well known. I may write more on this in the near future.

Another Take

My friend Ron Lake notes that he has a highly productive environment to build such large custom applications using NO-SQL. While I personally would start with conventional SQL tech, I have zero doubt Ron can also make it work with his magic stuff. He says:

AN ALTERNATIVE TO INTEGRATING COTS (AND THEN BEARING THE ONGOING COSTS OF INTEGRATION AND LICENSE CONFLICTS) AND CUSTOM SOFTWARE FROM THE GROUND UP IS TO USE AN APPLICATION ACCELERATION PLATFORM, THE OGC REGISTRY SERVICE. HERE IS A DESCRIPTION: [HTTP://WWW.GALDOSINC.COM/ARCHIVES/1699](http://www.galdosinc.com/archives/1699)

It saves time because:

- Its NO-SQL data model is intuitive and avoids the mapping and remapping of schema migration as your business information model evolves.
- Its NO-SQL data model contains higher level constructs like taxonomies, geography, associations than working at a relational or geo-relational level.
- It is a SOA web service out of the box, with a well-defined API which supports queries and transactions.

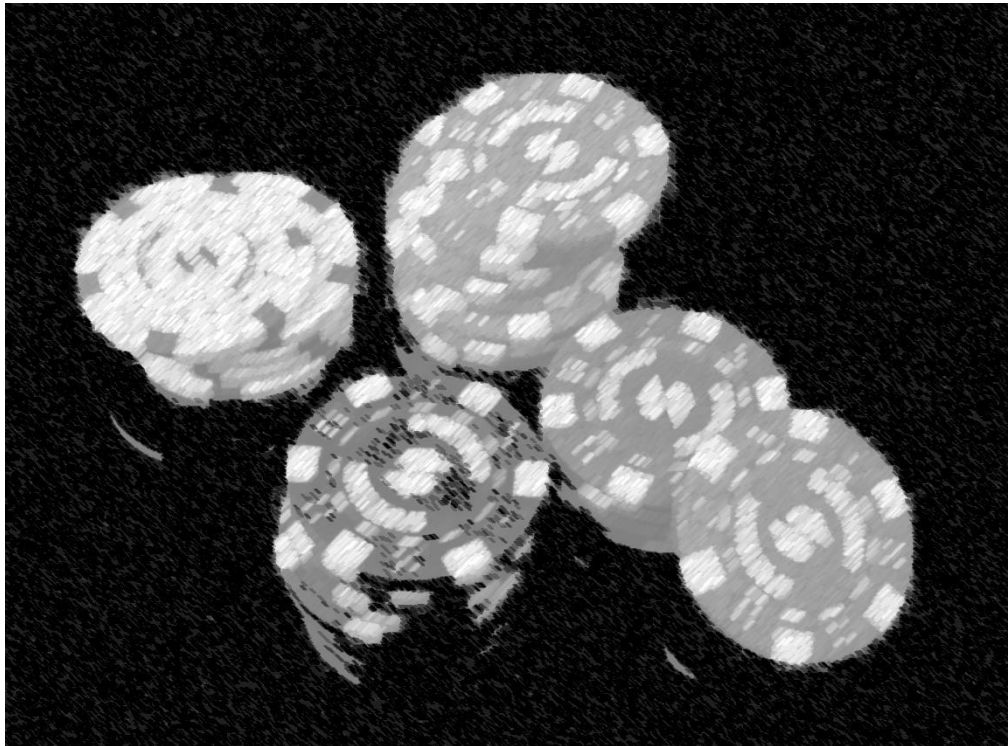
Conclusion

I suggest that we might stop treating large and complex application suites as radioactive waste. Custom development can build applications that better fit your proprietary business processes, and can provide improved organizational performance if applied to the support of mission-centered functions (not support functions), and combined with process re-engineering methods. In a realm where the technology changes daily, the truths you rely on may be obsolete. Some rethinking may be warranted.

Some References

- [*http://www.amazon.com/Data-Model-Resource-Book-Vol/dp/0471380237*](http://www.amazon.com/Data-Model-Resource-Book-Vol/dp/0471380237)
- [*http://www.amazon.com/Data-Model-Resource-Book-Vol/dp/0471353485/ref=pd_sim_b_1?ie=UTF8&refRID=0K95JP8GMR00X1K52JBE*](http://www.amazon.com/Data-Model-Resource-Book-Vol/dp/0471353485/ref=pd_sim_b_1?ie=UTF8&refRID=0K95JP8GMR00X1K52JBE)
- [*http://www.amazon.com/Data-Model-Resource-Book-Vol/dp/0470178450/ref=pd_sim_b_2?ie=UTF8&refRID=06FT5A3SN7HJDFB2G7ED*](http://www.amazon.com/Data-Model-Resource-Book-Vol/dp/0470178450/ref=pd_sim_b_2?ie=UTF8&refRID=06FT5A3SN7HJDFB2G7ED)
- [*http://www.amazon.com/Production-Workflow-Techniques-Frank-Leymann/dp/0130217530/ref=sr_1_1?s=books&ie=UTF8&qid=1405472584&sr=1-1&keywords=production+workflow*](http://www.amazon.com/Production-Workflow-Techniques-Frank-Leymann/dp/0130217530/ref=sr_1_1?s=books&ie=UTF8&qid=1405472584&sr=1-1&keywords=production+workflow)

3.13 SOLUTION ARCHITECTURE FRAMEWORK, NEVER POSTED



To perform solution architecture in a repeatable fashion, you will need a framework. When immature and not repeatable, no framework may be required in your practice. When your practice matures to become repeatable, you will soon need to write down your processes and methods, so you can repeat them.

Due to the confusion in enterprise architecture, with both a broad meaning and a narrow meaning to the term, some enterprise architecture frameworks are suitable for use in solution architecture. Other, additional frameworks have been suggested for solution architecture, but these are mainly usable only for software development. If you have systems that do not involve software, or involve it only peripherally, these may not be suitable for your use.

Here is a list of only a few possible frameworks you may choose to use for your solution architecture practice, with notes on my opinions regarding suitability:

DODAF

The Department of Defense Architecture Framework is suitable for use in solution architecture. It is fully capable of documenting non-software solutions. It is my personal preference for documentation of solution architecture.

This advice, and my recommendation, applies also to MODAF and other related frameworks from allied countries. They have been developed in collaboration, taking the best from thinking in various countries.

FEAF 1.X AND 2.X

FEAF 1.x contained little guidance for artifact type and construction. While I draw heavily on it for enterprise level architecture, I do not recommend it at the solution level. FEAF 2.x is completely different, and in combination with the Common Approach to Enterprise Architecture published by OMB, it provides an extensive list of artifact types and some guidance on construction.

Unfortunately these sources are mainly usable only for software solutions. Indeed you could say the new FEAF is specialized for only cloud based software solutions. I only recommend FEAF 2.x for government institutions without non-software solutions under management.

TOGAF

TOGAF is huge. Solution architecture is in there, but so is everything else. TOGAF in some sense views all the five activities of enterprise architecture as an undifferentiated mass in v9. Extracting out which of the 900 pages you will apply will be a difficult task, so using it as the basis for process maturity is possible but requires effort. TOGAF does contain aspects that can be used for non-software architectures.

RUP

IBM's Rational Unified Process is often listed as a proper framework for Enterprise Architecture. It does not meet the needs of enterprise level architecture as describe in these volumes, and should not be used there. It does meet the needs of software solution architecture. It does not have many features directed toward creating non-software solutions, and should be avoided for that purpose.

UML

UML is a means to document object oriented software solutions. It has some limited use outside software. I recommend that you use DODAF, which allows the options for relevant artifacts to be produced in UML. In this way you can document both software centric and non-software solutions, side by side.

ZACHMAN

At one time there was the Zachman Framework. Now it is the Zachman Ontology. You can analyze and document anything using Zachman's work. Anything. However it gives no guidance on visual artifacts, and is not used for those which are seen as transient. There is truth in that, but as-built and construction drawings must still exist. I recommend that all architects use Zachman's work for analysis and for metadata. Use it with your selected framework.

OTHER FRAMEWORKS

Avoid frameworks that charge you money. There are free frameworks to be had. One of the main advantages of a framework and standardized artifacts is that others can read and interpret them, so avoid frameworks with tiny circulation. Avoid proprietary frameworks that tie you to a vendor. Other than that, have fun.

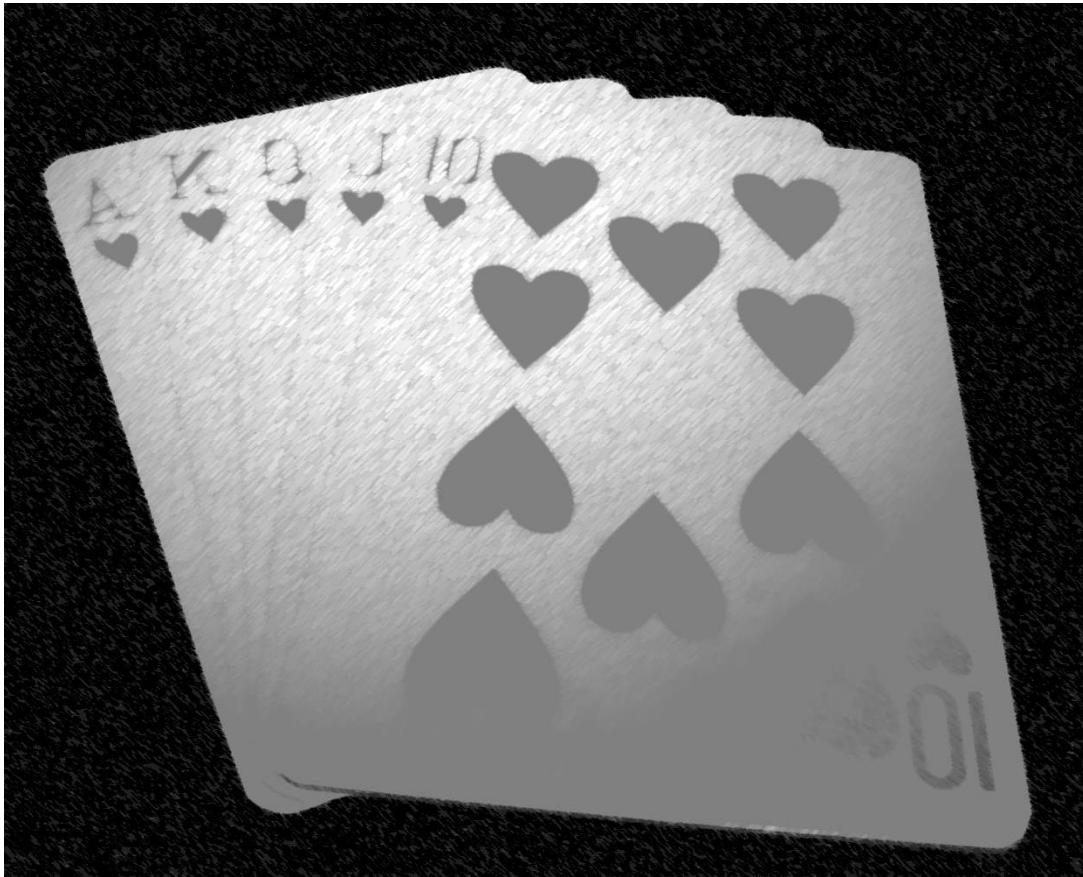
You can create your own framework as a starting point for your processes and eventual maturity management of those processes. In the early days everyone did this. Now there is little need for the additional work

CONCLUSION

If you pick a framework as your starting point for documented process, formats and methods of solution architecture, you can start producing standardized exchangeable artifacts immediately. This need not be the same framework as the one you choose for the enterprise level architecture, as needs are different. Regardless of what you pick, put the framework (once written or accepted) under continuous improvement and review per the maturity management process. Customize it to fit your organizational needs and practices, and measure its effectiveness.

Do not pick a software only framework if you produce solutions that have wider scope.

3.14 SYSTEM DEVELOPMENT LIFECYCLE, NEVER POSTED



In solution architecture there are many candidate system development lifecycles. Do not be fooled into adopting a software development lifecycle unless your shop only manages software solutions. Some candidates for your SDLC are listed below. I have added my opinion as comments on suitability.

ADF

The TOGAF ADF (The Open Group Architecture Framework – Architecture Development Methodology) started out as the TAFIM ADF (Total Architecture Framework for Information Management – Application Development Method). It was very effective in that role, perhaps more universally applicable than its present role. You can use the ADM to sequence activities.

DHS SDLC

The Department of Homeland Security has a System Development Lifecycle. Version 2 was very useful, though it contained some oddities (Stages 1-9 were preceded by Stage A, for example). Recent versions incorporate highly modified Agile methodology, with mixed results. It is at the cutting edge of modifying Agile and DevOps for use in the enterprise. You may want to see how that shakes out before adopting it. If you can get a copy, it is free.

DOD SDLC

DOD has so fully merges system engineering into acquisition that there is no distinct system lifecycle. If you are in DOD use their method, if you adopt their acquisition lifecycle you might use their method, otherwise skip it.

FAA SDLC

FAA has a perfectly good system development lifecycle as I understand it. I have not used it recently, so I cannot recommend it. If you evaluate free SDLC's for use, you should probably include this one. They do lots of things that are not just software. If you can get a copy, it is free.

NASA SDLC

Like FAA, NASA has an SDLC. I have not used it, but it is free as a result of being a product of the government. If you can get a copy, it is free.

INCOSE

INCOSE advocates the SIMILAR process. I have not used it, but documentation is free.

<http://www.incose.org/AboutSE/WhatIsSE>

ITIL

ITIL™ is not often thought of as an SDLC, but it is a lifecycle. You need a lifecycle but not 3 or four. If you adopt ITIL™ you have adopted a lifecycle.

PMBOK

PMI's PMBOK™ is a lifecycle, describing the order of many of the same documents as in any other SDLC. If you adopt the PMBOK as your sequence you may need to add to it, but it will be your basis.

SCRUM

Scrum, Agile, and DevOPS have built in assumptions of sequence. These are trivialized, and governance or stage-gate activity is minimalized. This may not work for organizations with oversight that requires proof that the software efforts have payback. It may not work for large or complex systems that require significant analysis. However it may work well for software only organizations that produce "apps" or small software applications with predetermined or minimal backend design

OTHER SDLCS

There are many other free SDLCs in academic literature and from other sources. Please do not pay for an SDLC. Do not use an SDLC for software only unless all your solutions are also software only.

SECTION 4: EXAMPLES:

Section 4: Examples:	116
3.1 A Nationwide Sensor System, December 26, 2014	117
4.2 A Global Nuclear Detection System, January 11, 2015	119
4.3 All Hazards vs Disaster Response, March 22, 2015	122
4.4 Geospatial Routing: EDXL-DE & CoT, June 30, 2015	127
4.5 EDXL is NOT ENOUGH, June 6, 2015	129
4.6 Mobile Network Architecture, Jul 28, 2015	131
4.7 IoT and Web Services	134
4.8 Building a Global Grid for IoT, June 27, 2015	136
4.9 Asynchronous and Synchronous Messages in the IoT, June 27, 2015	138
4.10 IoT Device Classes	140
4.11 Cyberspace	144
4.12 A Legacy System Upgrade, December 24, 2014	146
4.13 Architecture Lessons of the CHAPS Failure, October 20, 2014	148
4.14 Defeating DDOS, December 24, 2014	151
4.15 Advanced Security from the Trash Bin, October 15, 2014	154
4.15 Crypto and the Multiplexer Array, June 17, 2015	156
4.17 A Missing Unix Network Service, June 17, 2015	158
4.18 Another Access Control Model, June 6, 2015	161
4.19 Security Challenge-Response Example, JUNE 25, 2015	163
4.20 How to do Architecture: X-Browser, July 2, 2015	164

Most of the example posts are self-explanatory. One exception: 4.3 through 4.11 relate to and support 4.2.

Questions for Section 4:

1. Which of these examples seem like reasonable solution architecture or having elements of reasonable solution architecture? Which do not?
2. How do 4.3 to 4.11 relate to 4.2?
3. Could you build any of these in a better way? How?
4. How many tries at the same solution category does it take before your architecture is pretty good?
5. How do these relate to design patterns? Why did I not present a list of design patterns, like everyone else?

3.1 A NATIONWIDE SENSOR SYSTEM, DECEMBER 26, 2014



My first job out of college was to produce a nationwide sensor system. Arbitron Ratings Company, a subsidiary of Control Data Corporation at the time, desired to expand to radio monitoring and integrate this with its automated television monitoring equipment. The business produced independent assessment of show viewership by which sponsors of commercials would pay for air time.

EXISTING SITUATION

A statistically represented subset of houses would be monitored by agreement. Fees were paid to the household. In the existing regime, modified set-top boxes each established a modem conversation at night to transfer collected data to a set of Data General mainframes/superminicomputers. Data on TV viewing was collected all day long. A new telephone line was installed in each selected household to transfer data without interfering with emergency calls, such as 911.

FUTURE SITUATION

Each modem connection was a problem. The costs of installed modem lines would exceed \$3M/year in the expanded scheme. Data for multiple devices would have to be consolidated and transferred to minimize costs. New sensor types would be required for radio.

SYSTEM ARCHITECTURE

This would be a massive nationwide system with many thousands of households required to accurately represent viewership in different ADIs (Areas of Dominant Influence).

For some time the DG head end had operated with fault tolerance. One computer could fail yet the data would still be collected and processed. Duplicate databases were updated in parallel, and a recovery mechanism existed.

A household hub would collect and store data, transferring it as a single set for the household.

Multiple sensor systems, including the existing but modified set top boxes, would report data to the hub.

Multiple communications means would be used between the sensor systems and hub, including wires, radio over electrical wiring, and wireless.

Novel means to share the customer phone line would be investigated.

COMPONENT PARTS

The following component parts were produced to build the desired architecture.

I designed a household hub using a 80c188 processor and peripherals. This had slots for 2 communications modules and a modem.

I designed a modem capable of detecting if a household member was trying to use the phone during a modem session. It would signal the hub which would relinquish communications. This would save the \$3 million per year.

I designed the local communications protocol between sensor systems and the hub. This used a modified HDLC for a low speed, low radio interference network in the home.

I designed the first new sensor system microcontroller board using Z80 processor and peripherals. It had a connection for 1 communications module and several internal connections to a sensor suite (set top box, modified radio, etc.)

Two other teams produced sensor system micro-controllers.

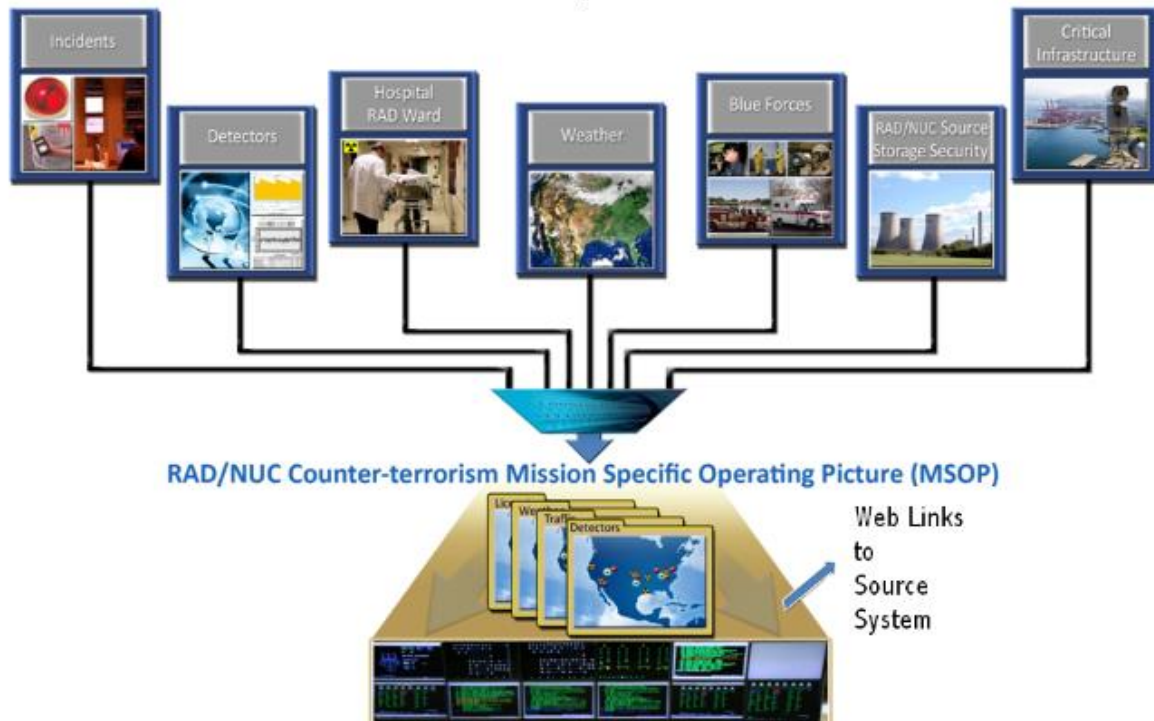
An external contractor produced the communications modules.

Suppliers produced modified set-top boxes.

DISPOSITION

CDC decided the effort was too costly and eventually sold Arbitron. The system was never fully deployed beyond a pilot market. Technology I produced resulted in 2 patents, and I was awarded \$50 for each. I have boards I designed and certificates from CDC for each patent on my wall, today.

4.2 A GLOBAL NUCLEAR DETECTION SYSTEM, JANUARY 11, 2015



The genesis of the solution eventually came from a book called "Power to the Edge" (David S. Alberts & Richard E. Hayes). It describes command and control in an environment where units operate independently, not at the behest of a unique central commander. This describes the federated democracy of the USA and civilian government quite well, where different jurisdictions have different money, different leaders, different laws and policies. To make it work, you share situational awareness across all those units. In practice, the majority of the units have no security clearances, so the situational awareness you share cannot be classified or suddenly no one can see it. We are speaking of, therefore, the system to share unclassified situational awareness.

It may have been the shortest interview ever. I had been pre-vetted, cherry-picked. I would determine how to connect rad/nuc sensors with state and local information systems. Six minutes flat.

Over the next eight years or so I would help DHS to establish three major programs related to information sharing of situation awareness data during disaster. This would include IPAWS-Open, UCIDS and MCM. The least involvement would be IPAWS, where I briefed Dave Waddell on what FEMA, at the time the customer, should tell S&T that we needed: "We do not need another operation center, we do not need another information system; we need something to connect all of those."

The greatest involvement would be MCM, where I created the concepts and architecture. The equipment was tested by DTRA to be capable. The prototype was tested by DTRA to demonstrate proof of concept. The demonstration was shown to PM-ISE, and declared a victory. However, a competing contractor cut me out so they could appear to be the leaders. (They subsequently fouled it up. Typical government contracting.)

I had help. Col. Stephen Hoogasian toured the inter-agency with me. Dr. Geoff Abbott provided introductions. Many more assisted. Even SECAF pitched in a bit. All agreed that constructing civilian situation awareness was very important in responding to another 9/11, or even a hurricane.

However DNDO, the sponsoring agency, decided it was not involved with counter-terror operations. They would just improve detector technology. Subsequently you might have heard much about the GNDA (Global Nuclear Detection Architecture) which tells us where to put detectors, but little about the GNDS (Global Nuclear Detection System) envisioned by the founders of DNDO. Situation awareness supports real operations.

Paul Agosta, Chuck Frawley and I then wrote a thesis about how to take the previous three architectures and create a superior hybrid, using the best features of each. This has been widely circulated.

Some of the basic ideas are these:

- The detectors are connected to detection systems at operations centers. The connection from the detector to the system was not the key element. This was handled.
- The communications between the eight types of operation system were the challenge. They needed to share situation awareness.
- Each operations center type had different sorts of software, and any one may be from one of several vendors. Yet these had to interoperate.
- The scale was massive, worldwide.
- The information to be shared varied by jurisdiction and operational state. Each had different laws and policies, and had to control its sharing for compliance.
- Various parties would share more in an emergency than without one. Privacy takes precedence in normal operations.
- No one is "in charge" of everything. Many leaders had to cooperate.
- People in the disaster area needed information on their disaster. People far outside the area needed information on other activities relevant to them. The operational "noise" of everyone hearing everything had to be eliminated.
- Information would be shared by a failure resistant, redundant distributed grid using fast network appliances with hardware acceleration, thus producing a far more cost effective approach.
- The mission became all-hazards situation awareness, not just CBRNE. Any classified bits were not part of this information sharing system.

BY COMPARISON DOD C4 IS A CAKEWALK.

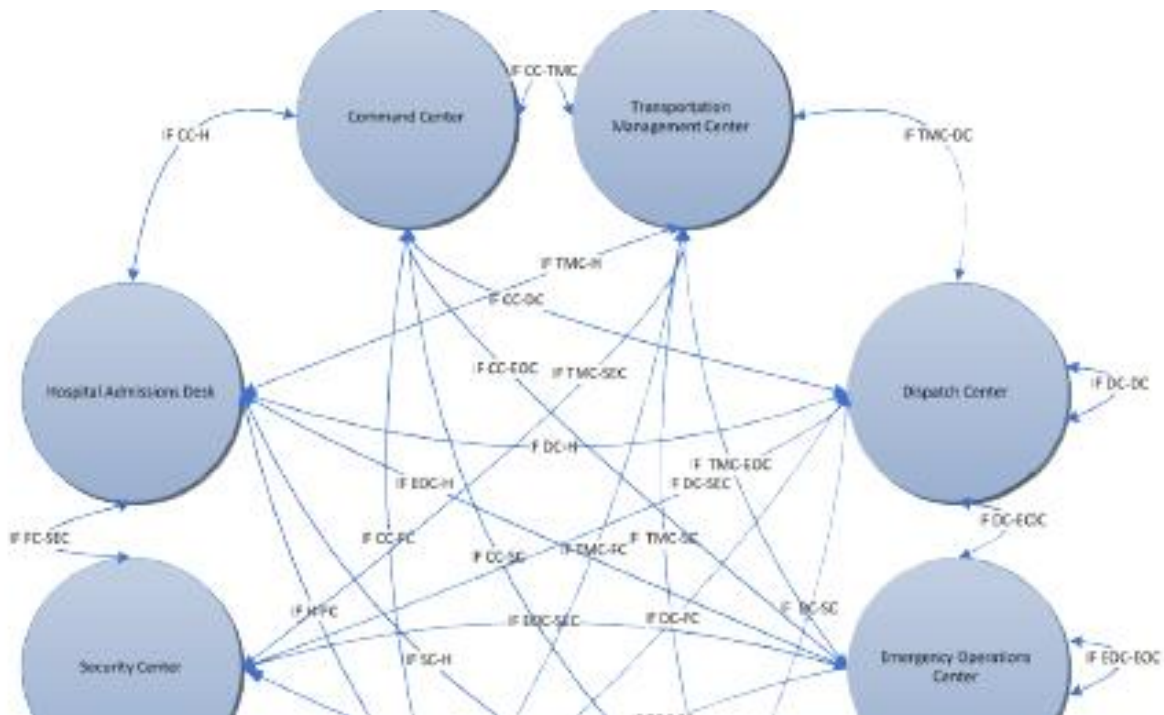
So there it sits, ready to save money versus comparable approaches, improve DHS operational performance, and save lives. But politics prevails. I await the shift of circumstances to improve the situation. (Actually nearly every country needs a system like this to manage disaster.)

Here is the key point: To make all those state, municipal, county, tribal, territorial and federal systems and organizations work together the key is common two-way institutional awareness. Until we have that, the world is not a safer place yet. This is the primary data sharing Congress and the 9/11 commission was speaking of.

An Enterprise Architecture Practitioner's Notes: Volume 3 Solution Level Architecture

(For the student of architecture this is still solution architecture. It may be global, cross organizational, and massive- but it is still solution architecture.)

4.3 ALL HAZARDS VS DISASTER RESPONSE, MARCH 22, 2015



Everybody is jumping in to disaster response. Whee! It is a market heating up! However significant work in this area has been ignored, and the solutions being produced in this area are inferior in scope and vision.

That is a pretty bold statement. You may wonder if I can back that up. It's a fair question. I have performed enterprise architecture at DHS, FEMA and DND. I worked wilderness search and rescue as a boy in the Pennsylvania Wing Civil Air Patrol Ranger program. I have an extensive background in logistics systems, including worldwide logistics systems, and logistics is the backbone of disaster response. But mainly I helped write a group thesis in this topic with two top notch DoD architects, it was reviewed by real PhDs. Compare that to all those other pundits, and you may agree that you should read these few words I have posted here.

TARGETING THE PROBLEM:

Let me make three initial points:

- We do not need more operations centers. Over 300 operations centers were created after 9/11, and we still struggle to find things to do with those.
- We do not need more disaster information systems. Hundreds of new disaster information systems were created or sold after 9/11. We have enough.
- What we do need is a means to interconnect all those, to share information between them. The 9/11 commission said so. The GAO said so. Congress said so. But not enough has been done.

To this end I struggled for 8 years and more to make headway at DHS. I helped to create IPAWS & IPAWS-Open, the main systems at FEMA to share disaster information. I helped to initiate UICDS at

DHS S&T, expressing that need described in the 3 points above. I produced the best architecture of the three for DNDO, to share disaster information across jurisdictions- but the mission changed there after pilots, prototypes and tests, and it was never fielded at scale. Then I helped write the thesis to document lessons learned and synthesize a way forward. That's more research and experience than most you will hear from.

DESCRIBING THE PROBLEM:

Let's talk about focus a bit, because disaster recovery alone does not serve the need. Honest. Listen up:

Natural disasters are only part of the problem. There is also terrorism. Disasters may be man-made or natural, and our proposed communications system needs to support the full range. DHS calls it "All Hazards".

Waiting until after the disaster happens is not the right approach. Before a man-made disaster, we have interdiction to stop the disaster from ever happening. Stopping potential millions from dying is a good idea. Before a natural disaster, we also have preparation and operations like evacuation. Supporting those is also a good idea. Disasters have a lifecycle.

Disasters come in all sizes, from local to national in scale and support. All big disasters start as local response and change over time. Addressing all those different scales is a good idea.

There are many different kinds of operations centers, each using different legacy software appropriate to different missions and focus. We need to interconnect across all those operations centers and systems.

There are many applicable data standards and message standards, not just CAP (Common Alerting Protocol) or roll-your-own SOA. Any real solution needs to support all those standards for interoperation.

Unlike DoD our democracy and civil government is not a unified hierarchy, and should not become a dictatorship in a disaster except as a last resort. (Any real solution must understand jurisdictions, and legal limits of power. A book called *Power to the Edge* describes methods that work in a federated democracy, preserving freedom and autonomy and improving speed of decisions in operations.) Any real solution will need to address this.

Response is real-time. A replication or batch update system to share disaster information will not work for all scenarios, and low latency is an advantage. One good example is aiding the one cop on the highway who is holding the terrorist with a nuclear bomb - getting there quick and rendering aid is a good idea.

Any real solution must be resilient, fault tolerant, able to function if a part of it is destroyed, in other words it needs to be distributed not centralized. No single points of failure should be present.

Federal backing is required to get everyone to use one such backbone for information sharing in disasters, rather than creating islands of information and silos of dysfunction. We need a single, unified, national backbone or clearinghouse.

OPSTEMPO or operational rhythm changes. Rules change as disasters escalate. Any real solution must allow changing rules as escalation occurs. Also those rules vary by jurisdiction and by body of law, and each jurisdiction needs to make their own decisions on when they can legally share what.

INTERCONNECTING OPERATIONS CENTERS:

Are you with me so far? Do you see why the present crop of solutions proposed are partial at best? If so, Good! We have hope! Now let's talk briefly about the eight categories of operations center that need to be interconnected for all hazards response (including disaster management):

- **Dispatch Centers** coordinate the work of police, fire and ambulance. Squint your eyes and take the blinders off, and they also coordinate water and sewer repair, electrical repair and waste removal. This is where much of the real work gets coordinated in disaster recovery, interdiction and such. These places use "computer aided dispatch" software from vendors like Intergraph.
- **Command and Control Centers** are military or pseudo-military operations centers (like FBI) that manage the orderly work of hierarchical groups. These folks play a big role in disasters, especially the National Guard. These centers use C2 or C4ISR software from vendors like Northrup Grumman.
- **Transportation Management Centers** were created by DOT many years ago to coordinate disasters in transportation systems, like highways, waterways and we will include air travel for our definition. These places use transportation management software and traffic management software and harbor management software from a variety of vendors.
- **Emergency Operations Centers** are usually empty, lights out, until a disaster - or have a skeleton crew. When a disaster happens people from many organizations flock in. They use software like WebEOC(tm).
- **Fusion Centers** were created after 9/11 to merge intelligence from state and local sources with federal sources, and distribute to local authorities. They use a variety of intelligence products.
- **Counterterrorism Centers or Sensor Centers** are a unique breed of management for vast groups of sensors such as CBRN and cameras. They may be co-located or alone, and use sensor management software of several kinds.
- **Security Centers** receive alarms from security systems and coordinate with Dispatch. They often also have a call center and call center software.
- **Hospitals** are not operations centers- but in a disaster they act like operations centers coordinating health services. Their role in coordination is crucial as a source and receiver of information. The admissions systems of hospitals can provide key information on which beds are available and which are filled, so we do not deliver patients to hospitals that cannot accept them as in Hurricane Rita.

These are the operations center types or categories we used in the thesis, and each plays a key role in information sharing. Each can also benefit from wall displays showing combined situational awareness from the combination of sources. In this way each may make decisions that complement the overall effort, as described in "Power to the Edge". Any real solution will include all these centers and systems in its scope.

COMMUNICATIONS PROTOCOLS:

We need to cover one last topic. In order to ensure interoperability and avoid paying the college tuitions of consultants children across the country, we will need to use standard communications protocols rather than a "roll your own" SOA approach. By using standards, any vendor of software will be able to produce adapters from their system (of any of the various types in any of the various operations center types) to the backbone communications system. Such a standards-based approach will allow wider development participation and far lower costs. All the following communications standards should be supported, as well as translation between them when appropriate:

- EDXL-DE is a communications wrapper that allows disaster information to be targeted to those working on that disaster, and prevents information overload of those working on other disasters.
- EDXL-CAP for alerts about disasters
- EDXL-HAVE assists in prevention of delivery of patients to hospitals that are full.
- EDXL-RM assists in loaning resources (firetrucks and such) between counties.
- EDXL-SitRep allows exchange of textual situational reports.
- EDXL-TEP allows tracking of emergency patients.
- IEEE 1512 base for transportation management messages between operations centers.
- IEEE 1512.1 for traffic incident information.
- IEEE 1512.2 for public safety messages.
- IEEE 1512.3 for HAZMAT messages (Note: state and local refer to CBRN as HAZMAT)
- NIEM N25 for CBRN sensor system control
- NIEM SITREP for textual situation reports by NIEM compliant systems
- NIEM SAR for suspicious activity reports
- NIEM APCO ASAP for security alarm communications
- IETF SNMP to allow debugging and administration of the communications network
- USAF Cursor on Target, the most widely used XML situational awareness protocol, including types as modified for FEMA and DR units a few years ago.

I hope this has informed and enlightened you, and even more I hope it will help you hold public officials accountable for creating a working solution to interconnect all this. Lives are at stake. Moreover, the solution is inexpensive and available off-the-shelf via equipment with massively better cost/performance ratios than what is being used now. Nationwide solution can be had easily within the existing budgets of some of the existing bloated ESB based systems.

The Thesis:

<http://www.unauthorizedprogress.com/images/A Reference Architecture for Geospatial Situation Awareness in All Hazards Response corrected .pdf>

INFORMATION SHARING OBSTRUCTION:

After 9/11 the Congress, the GAO and the 9/11 commission all issued findings that we need to share more information between government activities. While efforts focused on intelligence, information sharing in disasters is equally important. In Hurricane Ruth, helicopters brought

victims to hospitals that were full due to a lack of information sharing, and in 9/11 itself they were grounded on the New Jersey side of the river unable to help due to a lack of information sharing.

Yet some stand to impede information sharing efforts, even today. These obstructionists have had marked effect, and information sharing success has been limited. Speaking from the perspective of all-hazards (disaster and terrorist incident) response, let me answer these objectivists here:

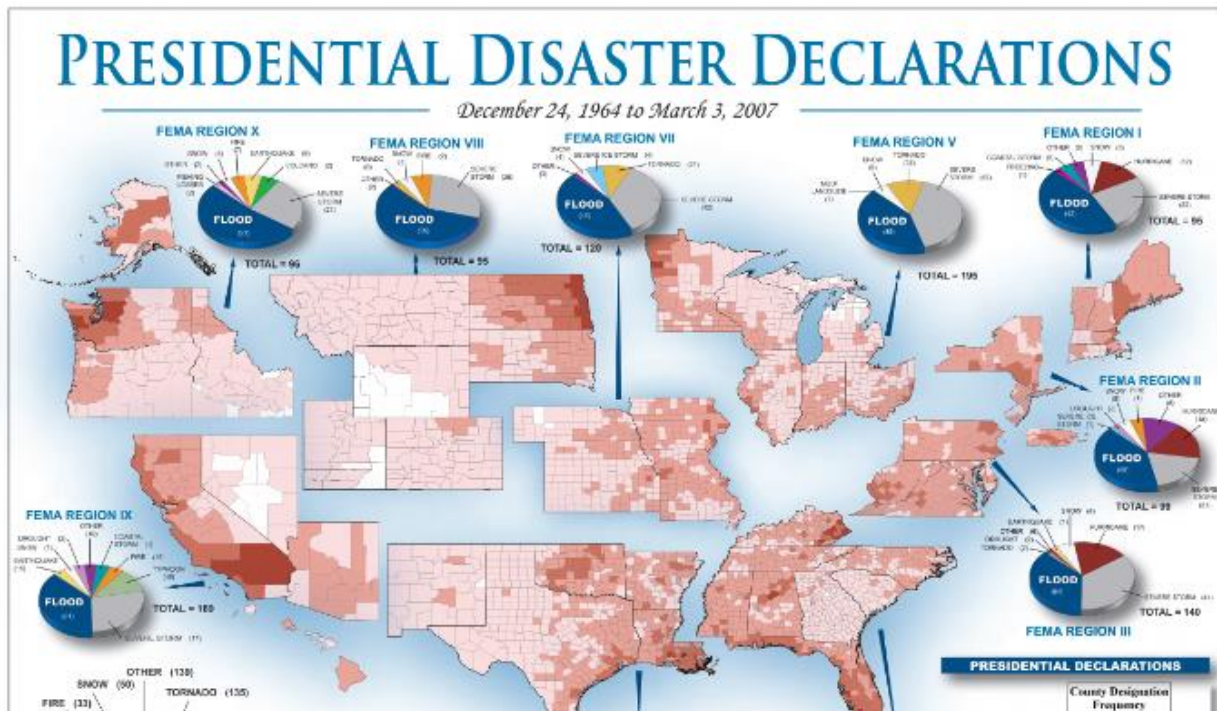
Taking police, fire and ambulance individually (as is the norm in this domain) there are about 100,000 jurisdictions in the USA. (59ish states and territories, 3,000 counties, 30,000 cities, towns and municipalities). If some small portion (say 5,000) do not wish to exchange information, surely the vast mass of others need not wait for these technologically backwards holdouts?

Exchange of information is becoming more popular due to inexpensive products produced by SAIC and Dr. Jim Morentz (Xchange Core). These areas find benefit in enhancing safety of citizens. Who are you to stop them?

In the "Securing the Cities" program NYC internal police office agendas, a self-serving consultant, a vendor protecting sales, and some weak kneed Federal officials achieved failure in implementing "defense in depth" around NYC to enhance defense against terrorism. Millions tonight in Connecticut, New Jersey, Eastern Pennsylvania and Suburban New York may sleep less soundly knowing that a bit of cowardice, greed and self-interest prevailed. Later efforts stalled due to lack of success in NYC. This is an example, the poster child, of delaying information sharing for social reasons. Millions were wasted.

Having been involved myself in disaster management I will relate to you that I once stood in a trailer park at 3 am surrounded by parts of people mixed with aluminum siding, plywood and other materials. Many had died due the odd attraction of aircraft to trailer parks. If ever a disaster occurs that should have been avoided by information sharing, and the bodies pile up, those folks standing and looking at the dead may think of you obstructionists.

4.4 GEOSPATIAL ROUTING: EDXL-DE & COT, JUNE 30, 2015



US Government Image, from FEMA.

What if you wanted to send information to an area, like an email to everyone in Detroit? No, not with a list of everyone in Detroit and some long transcription exercise, just directly do that?

Why would you want to route messages to people in a given area, or watching a given area? The identification of the need came from disaster management, especially in regard to notifying people of imminent disaster. "Hey you, go hide in the basement a tornado is coming!" That sort of message is very useful.

The concept has been extended to the future. "Hey you, a hurricane is coming tomorrow, prepare!" Obviously that is pretty useful as well.

But what if you are a disaster manager watching the area, not in it. Say you are the manager in charge of keeping people fed in your state. A disaster is going to strike the other side of that state, far from you. There is a notion of subscribing to an area distant from you that is also useful. You could also then subscribe to the area where you are, by the same mechanism.

However you may want to limit the kinds of messages you receive. Say you are a fireman, and do not want to know when a pocket is picked. You want only fire related messages for your city, not police. That would be useful.

A message header to allow this kind of messaging was created by OASIS, a standards body. The header and the standard are called EDXL-DE or Emergency Data Exchange Language, Distribution Element. It allows you to describe where your message should be distributed. You can distribute to a city, a part of a country, a polygon on the surface of the earth. You can limit distribution to police,

firemen, ambulance services, HAZMAT response, or whatever. You can name a disaster and send information to only people interested in that, like "Hurricane Matilda".

This is all somewhat revolutionary. We used to do something like it when the telephone company had exchanges that mapped to your neighborhood: you could dial everyone in you exchange and speak to all your neighbors if you liked. But the Internet uses TCP/IP which is not tied to geography. So the idea of geospatial routing arose.

The intent of EDXL-DE is to route all sorts of emergency messages. The most successful OASIS message format is CAP or Common Alerting Protocol, and all those disaster alert examples above are CAP type messages. There are a range of others.

However the most interesting type of message to me is from the USAF (developed by MITRE). It is a message type that really needs, and currently uses limited geospatial routing. The message format is called Cursor on Target or CoT. Yes, it was originally for blowing things up. However, it can be used for a wide range of other things. (Somewhere here I have a letter from SECAF saying we can all use it for humanitarian purposes) You can keep track of people, cars, trucks, fires, dogs, tornadoes, hurricanes, floods and volcanic plumes with it. You can issue moment by moment updates on position.

CoT is so well designed that you can say "Hey, I see a hazard and I do not know what kind it is yet". That is very helpful if you see vast rolling clouds of doom, but have not yet determined if a volcano did that. (Another MITRE message format claimed to do that, UCORE, but proved to only track the known named terrorist as he rode on the airplane and a few other tricks.)

In combination you could use these two standards to keep the nation aware of anything from natural disasters to enemy actions to terrorist activity to potential targets. It is exactly what DHS (the Department of Homeland Security) needs for its broad all-hazards mission. Unfortunately, the network to move such data between disparate systems in states, cities, counties and the Federal Government has not yet been built. (There was a study, a pilot, a prototype, and then competitive bickering.) But I remain hopeful!

Such a system would not be useful only to the USA either. Any country could use one to unify all their systems and provide unified situational awareness, unified command and control. Heck, even Canada could use it.

'Someday, perhaps.

4.5 EDXL IS NOT ENOUGH, JUNE 6, 2015



OASIS has been paid by DHS to produce digital information protocols under the title Emergency Data Exchange Language. This is a wonderful thing. These protocols have been very helpful, very successful. Most successful of all is EDXL-CAP or Common Alerting Protocol.

However this set of system-to-system protocols is not enough. It is not nearly enough to cover the DHS mission, all emergency management needs, or all counter-terrorism. Other protocols must also be used.

Imagine such machine to machine protocols (M2M) as a vocabulary, in which you can form a dialog, No M2M protocol devised has defined the meaning (semantics) for all subjects everywhere. Humans speak of divisions of knowledge (Hayek) as evidenced in the Dewey Decimal System, for example, and each such division has its vocabulary. How could anyone think that machine communications would be otherwise? If government spent many millions, an all-encompassing system of M2M protocols would not result.

The EDXL of OASIS encompasses several different protocols, covering the following kinds of conversation:

- Distribution of emergency messages
- Trading resources between organizations
- Available hospital bed capacity by ward
- Alerts
- Situation Reports
- Tracking of patients

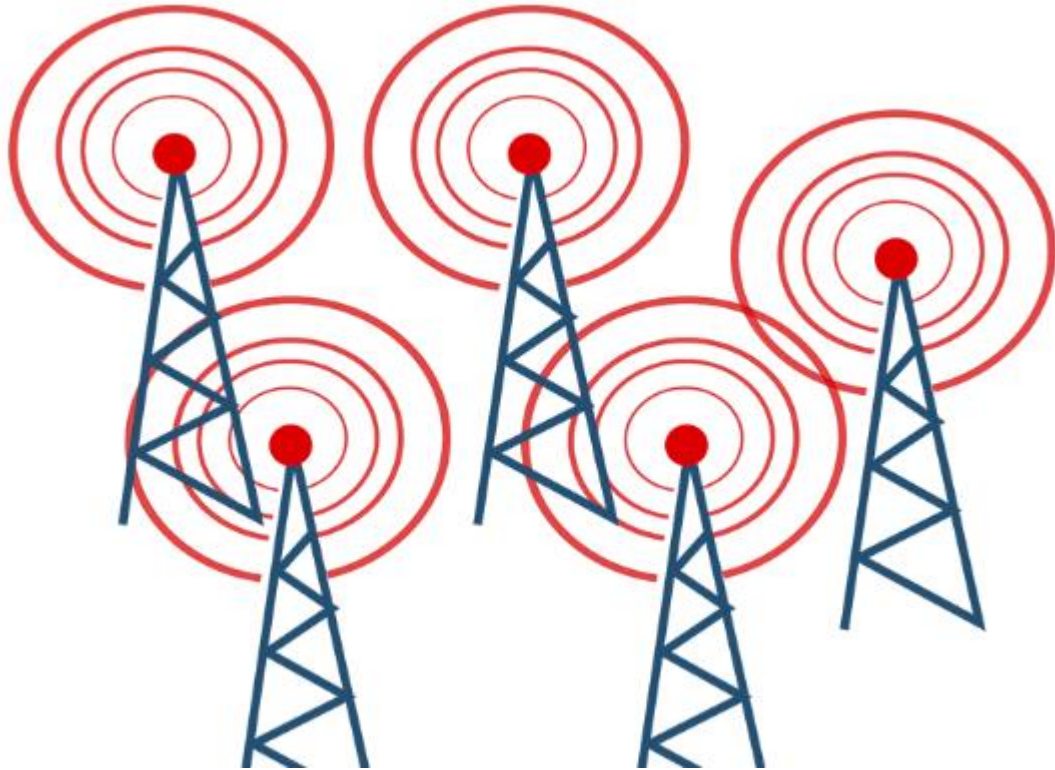
The following M2M conversation topics are not covered by OASIS EDXL, and perhaps should not ever be as someone else already standardized them:

- Geospatial situational awareness (Think of icons placed on a map)
- Traffic incidents
- HAZMAT
- Coordination between operations centers
- Alarms from security systems
- CBRN sensor information and validation of CBRN incidents
- Management of the emergency messaging network itself

It should be clear that OASIS alone does not, and will not be, the only protocol needed to communicate in a disaster or emergency. They do not have a monopoly, a lock on all disaster or emergency communications. Those vendors supporting OASIS EDXL protocols alone are supplying products inadequate to cover the full need.

Choose the protocols to fit the need; do not think of them as exclusive or competitive IP, or you will be doing a disservice to the customer community, and the disaster victim.

4.6 MOBILE NETWORK ARCHITECTURE, JUL 28, 2015



Radio range is a complex subject. Range varies with terrain, obstacles, frequency, weather, transmitter power, antenna gain, interference, and receiver sensitivity. In municipal or suburban settings it often reduces to "line of sight". Several years ago radios communicated mostly point to point, and maximum range was very significant. Shortwave transmitters still communicate from point to point across the globe. Business FM radios still find limits of miles between units. But cellphones are a different story.

CELLS

Cells were adopted because the spectrum is limited in size: there are only so many radio frequencies. It is more effective to use cells, at lower power, communicating from a mobile radio to a tower than to have all radios blasting at maximum power as far as they can. In this way the same frequency can be used a few miles away, out of range of the next user. Many more people can use every frequency. This strategy is used widely at higher frequencies, and less at lower frequencies where low power can still travel far away.

The limiting factor for cellular radio range is the cell size, the radius from the cell edge to the tower. Alternately you can view this as the distance between towers. It limits 3g cell phones, and because the tower was placed there and won't be moved it often limits 4g and 4g-LTE the same way in the same place. Once the signal hits the tower it is sent forward by wire or fiber, and radio is no

longer used. Hence, if both parties are in range of a local tower, cell phone calls can connect people worldwide.

SMALL CELLS

If small cells are good, smaller cells are better. Cellular companies now think in terms of macro-cells (the conventional legacy), microcells and picocells.

- Macro-cellular nets, with cell radius 1 - 30 Kilometers, or up to about 19 miles.
- Micro-cellular net, with cell radius 200 - 2000 meters, or perhaps a mile and a quarter.
- Pico-cellular nets, with cell radius 4 - 200 meters, 600 some feet.

The advantages of cellular radio architecture are magnified with smaller cells. More users are possible, more simultaneous conversations, more total throughput. Incidentally more bandwidth may occur, as higher frequencies allowed by shorter range can convey more data. (This higher data rate is part of what makes 4G better than 3G).

Bluetooth, by the way, can be seen as a very small cell centered on the individual or his device. Bluetooth is often said to have a range of 10 meters, or 32 feet. This is the range for a "class 2 device", and an alternate specification allows for ten times that range, about 100 meters, still within the range of pico-cells.

WI-FI

But Wi-Fi (IEEE 802.11) is now a functional replacement for cellular connection. Both phone calls and web access can function over either. The maximum range from a Wi-Fi endpoint to a Wi-Fi router is either 50 or 100 feet or about 17 to 33 meters. This is smaller than a pico cell. You still might think of it as a cell, and you might even call it a Femtocell. Bluetooth is even shorter in range: an Attocell? (prefixes for negative powers of 10 in groups of 1000s: milli, micro, nano, pico, femto, atto...)

A class of equipment has arisen to take a smaller cells in on one side and connect to bigger cells on the other. A Wi-Fi "Hot Spot" is such a device. So too is a cellular repeater, in a way.

BIG CELLS

But the benefits of the cellular approach need not be limited to 19 miles. Larger cells could exist. Legacy macro cells need not be connected by fiber or copper. You can connect these legacy cells by radio technologies. For instance you could connect cellular towers by microwave links, but little is stopping us from creating wider range cells of 100 miles or more. Let's call these mega-cells.

However as cells get larger, lower radio frequencies are desirable for propagation across longer distances. Lower frequencies can carry less data (assuming reasonable bandwidth). Methods like spread-spectrum may be used to enhance bandwidth and data rate, but have limits.

Evolution continues: IEEE Wimax will increase data throughput and increase maximum range to about 30 miles.

http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4215486&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxppls%2Fabs_all.jsp%3Farnumber%3D4215486

Also a long wavelength, low power, longer range, lower data rate standard is in development, Perhaps driven by the "Internet of Things".

<https://standards.ieee.org/findstds/standard/1902.1-2009.html>

On the other hand, the 80-20 rule as applied to networks indicates that 80% of traffic is local, reducing the traffic requirements of infrastructure at higher levels of architecture, This effect is lessened by broadcast media, centralized websites, cloud based data centers. So ultimately, at some point, radio must give way to fiber or copper wire.

But mega-cells may still be useful, to some extent, in hostile terrain and wilderness or in natural disaster. One type of mega-cell is the communications satellite, which has limited communications bandwidth available but can cover wide range. Directional antenna technology can increase throughput or bandwidth.

SOFTWARE

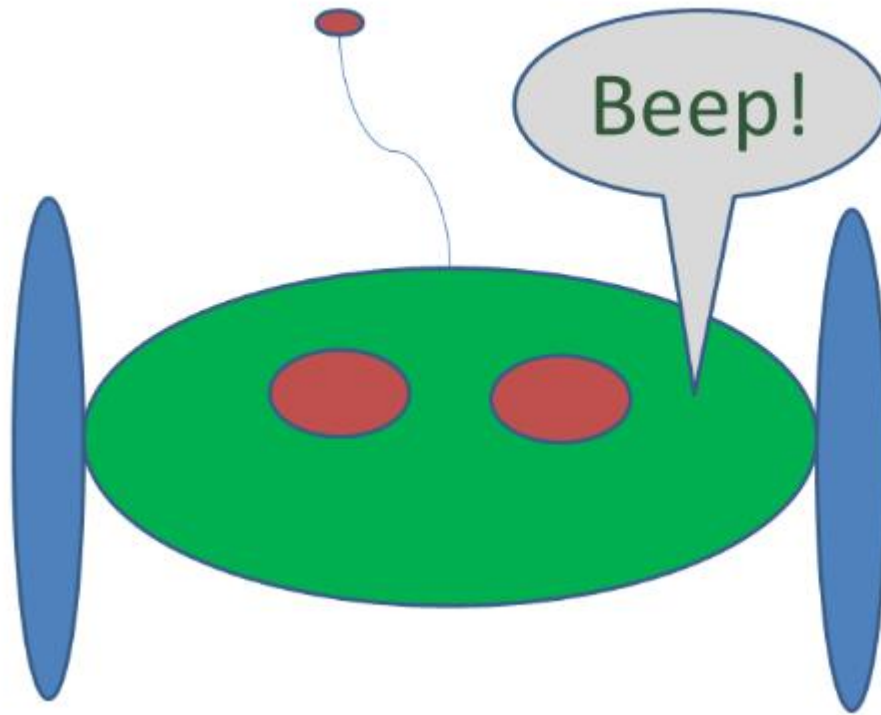
At one time we were very concerned by protocol efficiency. Protocols to transfer voice communications efficiently are "connection oriented" and those for data traffic are "packet oriented". Such concerns have partially faded into obscurity, and inefficiency rules for voice traffic. However cellular low level protocols retain quirks and features very different from Wi-Fi. These may be significant, but are often masked by TCP/IP in practice.

Many speak of "mobile architecture", but my cell phone runs the Android™ OS and so does my tablet. Android is a Linux-like variant, and desktops or servers often run Linux. The most significant difference in mobile devices is screen size. Secondly we may speak of processing power and storage/memory size. Code-monkeys would jump to say apps, but what are apps other than applications, limited by screen size, processing power and memory/storage. Apps will become larger applications again when big screens on wearable headsets overlaid on your vision come into fashion. Won't they? Maybe screens will become a neural link to the visual cortex. Who knows, but apps are a limited by screen size.

Skip screen size, as most devices will lack these, for the IoT. The Internet of Things will be populated by ubiquitous computers of small size, small power (small batteries and processors), small memory or storage, and small radio bandwidth. No notion of wire or fiber seems to piggyback on the IoT parade. But some such devices communicate with others of their kind (swarming and such) and some speak only to local devices such as thermostats or security systems. The risks of exposing the local cell of IoT to the broad Internet and threats there are extreme, and may not be matched by much benefit in some applications. There is a tradeoff, as in most of engineering. Perhaps, just perhaps, a non TCP/IP local protocol should be used for many IoT devices.

See also: <http://www.wirelesscommunication.nl/reference/video/bmrc/07linnartz.html>

4.7 IOT AND WEB SERVICES



I just read a nice article on DARPA, robots, disasters, warfare, & security. In there they spoke of web services. Nifty. I also recently read a white paper describing how the IoT is all about connecting little devices to web services, and fancy software abstractions. Neeto.

In the first article it described the operating environment at a disaster, lacking cellular connectivity. It also described the disaster environment at Fukushima, with zinc plates and rebar blocking radio. It did not seem to mention the increased noise at the radio detector due to radiation. Then it went on to talk about those web services. Radio won't get far in such a place, maybe a few dozen feet.

It reliably goes a few feet due to a phenomenon called burn through. I studied jamming in the USAF, and I know in war or conflict that enemies intentionally attempt to limit communications, including in concept your cellular connection to the internet for web services. (OK, I know a bit more than that.) But this phenomenon, burn through, allows a transmitter and receiver in proximity to communicate despite a more distant high power jammer. The effective power at the receiver, the signal, is affected by inverse distance, you see. So you can still talk to things near you.

My understanding of these various phenomenon is that your cellular connection from your IoT or robot device has about zero chance to reliably connect to your distant cloud based web service in various hostile environments. Nada. Null. Non. Not happening... Never send a software guy to do a systems job. This will also happen if your mobile IoT or robotic device moves out into the rural areas where there is no cell coverage. All these reference architectures emphasizing IoT, robotic

and mobile devices using distant web based cloud services have a flaw based in widely known physics and engineering.

Not that you will never have access to the web service, just that you cannot depend on it. Now one of the articles (with the DARPA guy) emphasized autonomy. Good stuff: that will help. When your robot cannot phone home for instructions it continues to operate in a limited way. But we need a sort of communication for short range communications - independent, autonomous device communications between local devices - because you can pretty much count on that. That needs to be a part of any, all, mobile, IoT and robotic reference architectures. Lacking this feature the architecture is a joke, flawed, without basis in ordinary reality.

Luckily all that technology exists. It is just not what the folks who write software and white papers these days are used to writing about. There are peer to peer mechanisms to allow one thingy (unprofessional word alarm, so substitute entity and keep reading) in our architecture to communicate with another as "agents". They do this using pre-arranged messages through choreography, with no central arbiter communications arrangements before the fact are called standards. Agents are the things that let automated marketplaces function, and that cause groups of previously not introduced software to coordinate. There are books on them, articles, papers.

Surely you have seen all that research about devices or robots cooperating in groups or swarms? Imagine the utility of one robot asking another about the local terrain map and getting an update that the wall collapsed over there. Yup, not all architects are stymied about what to do when your precious web services are not there.

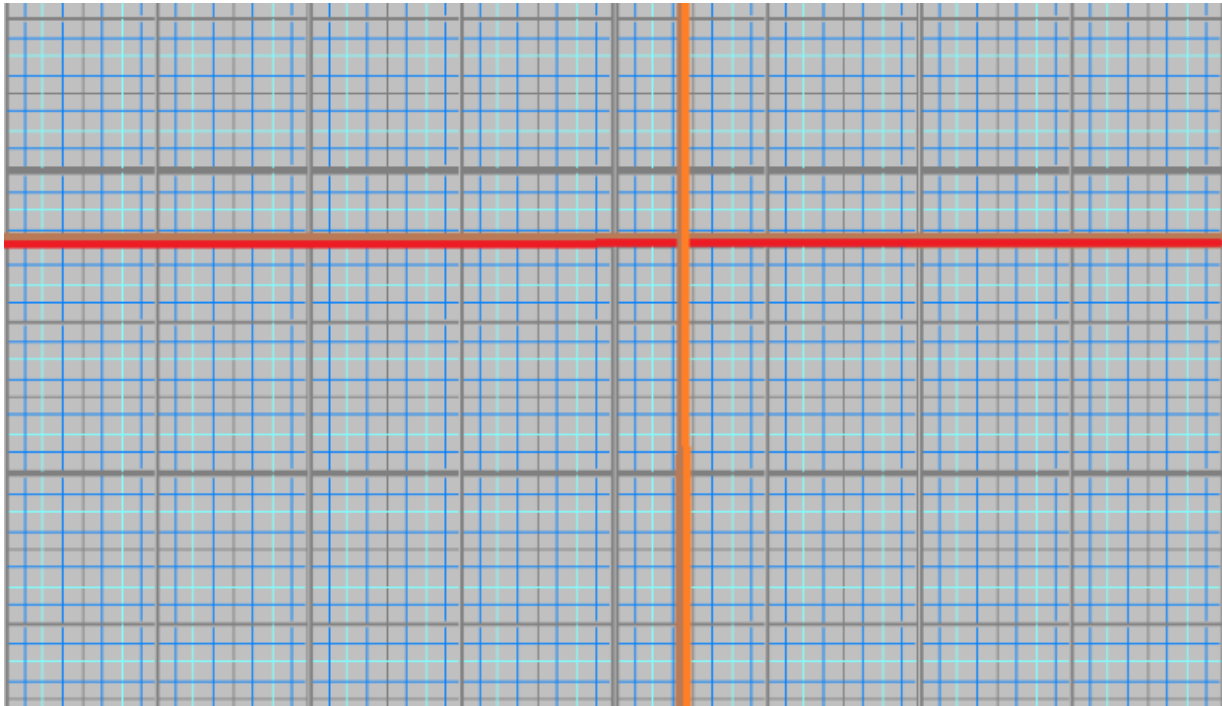
Even mechanisms to declare the smarter local thing in an unequal exchange to be in charge, a sort of local client/server communications, both exist and are common. Consider how your Bluetooth headset connects to your cell phone. No recourse to the World Wide Wonder Web for some unavailable service required. Not even in Nevada between towers, or in Pennsylvania behind the big hill.

The use of coding of messages more efficient than XML may also be required. Bandwidth or more precisely capacity in compromised radio environments is a tradeoff with reception and noise rejection. Yup, being good at IoT may require understanding encoding by bits and bytes, the old fashioned way. Luckily, most of this exists.

Also, in your robot/disaster/IoT/mobile/war architecture you should think about processing power, abstractions, and battery life. Security and safe operating systems with process isolation (like in a good RTOS), or avoiding virtual machine vulnerabilities, throwing in battery life and available storage, may be a good idea.

The current crop of architects may be inadequate to think about or envision proper trade-offs for such an architecture. Luckily some architects are already available to address these issues, with experience in them. These issues were common in the '70s and '80s.

4.8 BUILDING A GLOBAL GRID FOR IOT, JUNE 27, 2015



Suppose you desire to build a worldwide enterprise environment for the Internet of Things. How will you build it?

Let us suppose your local IoT devices have a local hub connected to the Internet, or are themselves connected. Imagine you have two alternatives for your global environment. In alternative A, you will use Web Services and Orchestration. In alternative B, you will also use choreography and message standards.

ALTERNATIVE A

Further details of alternative A would have global servers for different sorts of services. If you need a situation awareness service to report locations of important objects, or to ascertain what is around you, you contact the central global server for that type of service. If you need financial services or news or whatever, other central global servers provide that.

Several complications cloud such an architectural approach. First, each global server needs backup at an alternate site to protect the system from circumstances where the primary site is destroyed. This is especially critical for defense or disaster recovery applications. Second, a single central server may be cut off from various global locations, and regional copies are a good idea. How will you synchronize these? Third, different organizations will present political barriers to use of a single central anything. Multiple head ends are the norm, divided by organizational boundaries (Google vs Amazon would be an example).

For these reasons and others, a pure SOA Web Services architecture is an inferior approach. Reduced reliability, especially in adverse conditions, would result from such an approach.

ALTERNATIVE B

Alternative B combines multiple regional or redundant SOA servers into a grid. These servers communicate with their peers via choreography, messaging standards and asynchronous messages. Using these technologies each participant knows the rules and no central single point of failure is required in the system. If any server fails, the sensor or end nodes can pick another server and connect to it. This is basic communications engineering compounded with basic system engineering, and does not express new or radical engineering ideas.

Hence to build a more fault tolerant, more resilient, more recoverable global environment for the Internet of Things, you would interconnect services from regions or competing companies or organizations with different jurisdictions and leadership using choreography, messaging standards and asynchronous messages. Such engineering is built into other global systems such as the world banking networks, credit card networks and the stock markets. Unfortunately not all such global systems are as well engineered, and the DoD GIG could use some review.

The engineering principles involved do not change with technology introductions such as web services. They existed before web services, and will outlive them. The physics of information and the principles of building of systems are more fundamental.

FEDERATING ESBS

These results may be quite broadly applied, beyond the Internet of Things. For example, if your large organization has accumulated many ESBS and you require some means of interconnecting them, you might federate them using more ESBS creating yet more single points of failure in your architecture. Alternatively, you might interconnect them using choreography, data standards and asynchronous messaging in such a way as to mitigate even the current single points of failure.

LOOPHOLE

If the global IoT application in question simply records history, and any one portion of the system need not know the conditions at another (because of that one-way flow of data), you can avoid the issue entirely. You can simply use a large database to aggregate the history from many discreet servers after the fact.

Not all architects are capable of grasping issues at this scale, or have studied them. This particular issue can quickly separate those who simply regurgitate vendor claims from those who have studied the fundamental engineering involved by reading the texts in which such fundamentals are published. It takes some time, that study.

4.9 ASYNCHRONOUS AND SYNCHRONOUS MESSAGES IN THE IOT, JUNE 27, 2015



ASYNCHRONOUS MESSAGES

If the postman knocks on the door to give you the mail, that is asynchronous. You were interrupted from doing whatever and you received the mail when it arrived. In a computer system if the program is interrupted to receive the message exactly when it arrives, that is asynchronous.

SYNCHRONOUS MESSAGES

If you go to the front door at 11am every day and check the mailbox, that mail is synchronous messages. You asked for them, and you received what had shown up until that time.

CHOREOGRAPHY AND ORCHESTRATION

When a central authority directs or coordinates action, you have orchestration. If a policeman stands in the center of an intersection and directs traffic, it is orchestration. Orchestration is often associated with synchronous messages. Orchestration and synchronous messages are associated with local central authorities (governance) that determines message format (like in SOA Web Services).

However at an intersection where there are four stop-signs and everyone waits their turn without a policeman in sight, you have choreography. No central authority directs anything. Every participant knows the rules. Asynchronous messages and choreography are associated with standards for message format. Such technologies are also called "peer to peer".

AGENTS, SWARMING, SELF ORGANIZING SYSTEMS AND THE IOT

Systems that self-organize or swarm using agent technologies do so without a central organizing authority. These systems use asynchronous messaging and choreography based on standardized messages published so each participant knows the rules. The Internet of Things will largely be based on these types of technologies. They are more scalable and more portable than orchestrated systems.

EAI, ESB, SOA AND THE STATUS-QUO

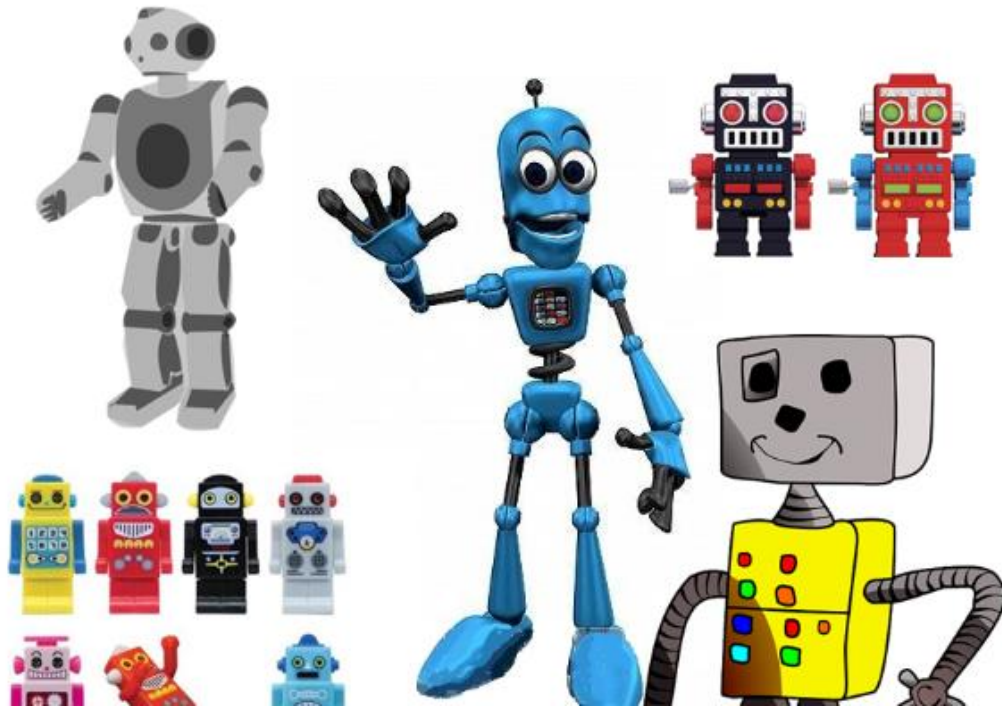
EAI systems, ESB systems, SOA and similar based on local governance determining messages for local use apply orchestration, where the SOA application server, the EAI server or the SOA server is the central authority. This is less scalable, less portable, than the alternative.

THE GREAT PENDULUM

The great pendulum of popular opinion has recently favored orchestration, a central server. Such opinion rarely reflects actual engineering tradeoffs. We can predict that choreography and asynchronous messages will again increase as the IoT matures, assuming that it will not be built with central authorities.

The engineering constraint driving all this is communications, and either range or bandwidth require more power, more batteries. Most IoT devices have small batteries, small processors and low power communications. Alternatives in such a design space include creating local hubs or concentrators as central authorities and spreading them everywhere the IoT will go. Such a central box would then have more power, perhaps a wall plug, and higher communications range or bandwidth. Where such a box is not present, the IoT devices dependent on it fail.

4.10 IOT DEVICE CLASSES



We do not want our cars accelerating and the brakes disabled by some distant person with a grudge. Nor do we want our pacemakers disabled. Less critical, our home security systems could be bypassed, or our HVAC set to 110 degrees Fahrenheit of heat in the bedroom. Our personal security, and that of our family, will be in the hands of the Internet of Things.

Autonomous devices in the Internet of Things may be characterized by their exposure footprint. As processors become ubiquitous a wide range of device profiles will coexist. Some will be far more reliable than others, others will be more easily exploited or disabled than some. Here is a quick analysis dividing these devices by class for exploitation, based on existing device characteristics.

CLASS A DEVICES

These devices will have a connection to the Internet, may receive software updates via the Internet, and may transmit sensor data and receive control data. They will often have a sophisticated user interface. They will "do it all." They will be unlikely to have a reset timer or an RTOS (Real Time Operating System) with process isolation and deterministic response. They are more likely to run a variant of Windows or UNIX/LINUX. They will normally receive software updates over the Internet and allow remote configuration via the Internet. As a result of these characteristics Class A devices will be highly vulnerable to exploitation. Such devices will commonly have apps to interface with local Class C, Class D and Class E devices via the local network (usually, wireless, Bluetooth).

CLASS B DEVICES

Class B devices will have an Internet connection and will receive software updates. They will be used to control a sophisticated device such as a refrigerator, range or HVAC system. They are unlikely to have local, non-Internet network connections. They may have hardware to perform a

variety of power management and error resistance functions. They will normally receive software updates over the Internet and allow remote configuration via the Internet. These devices will be vulnerable and exploitable.

CLASS C DEVICES

Class C devices will control other Class D and Class E devices. They will not have Internet connections and will not receive Internet software updates, but will often have a user interface and sophisticated software. They may be autonomous. These devices may be highly mobile, and may have extensive hardware and software for power management and error management. Designed to resist the exploitation footprint of a Class A device and for mobility, this class may often characterize autonomous robots, including toy robots.

CLASS D DEVICES

Class D devices will have sensors and actuators, and will perform actions beyond passive sensing. They will transmit sensor status and receive actuator commands. They will have a local network interface but no Internet interface, being mobile and lacking sufficient processing power to interact with the Internet themselves. They will probably have hardware and software to support process separation, time-out reset, power-down, process separation and deterministic response. No software updates will occur via the Internet. This class of devices will be highly resistant to exploitations, and will have minimum vulnerabilities. Most exploitation will occur via the Class A devices they are connected to.

(A Class D' device is possible that would have Internet connection, configuration via the Internet, updates via the Internet, and would have increased vulnerability and cost.)

CLASS E DEVICES

These devices are characterized by minimalism and power constraints. They will have a few sensors and occasionally broadcast their status, but will rarely receive network data other than network synchronization and status. They will have a local network connection but no internet connection, and will not receive software updates via the Internet. Hardware and software to shut off the processor, to reset the processor in the event of software error, and process isolation may be present. As a class these devices will have few vulnerabilities and will be exploitable mainly through the Class A devices they are connected to.

(A Class E' device is possible that would have Internet connection, configuration via the Internet, updates via the Internet, and would have increased vulnerability and cost.)

POWER MANAGEMENT HARDWARE

Common practice in hardware design includes hard means to perform a power shutdown, enter sleep or hibernate mode to conserve power. In mobile devices this may include shutting down networking, including wireless networking. Sometimes timer circuits are connected to hardware to perform a power up operation.

ERROR CONTROL HARDWARE AND SOFTWARE

In real-time systems sometimes a timer circuit will be wired to power cycle the device unless the timer is reset before triggering. Imagine a device that shuts the controller off every second, paired

with a software process that resets the device, scheduled to run every half second. If the operating system is hung the process does not run as scheduled and a power cycle and reset occurs.

Further RTOS (Real Time Operating Systems) are designed to isolate processes, so that when one software process fails the device continues to operate. MS Windows does not operate this way, and neither do most UNIX variants. As no user is there to reset the device and it may be required to perform operations like causing a heart to keep pumping or making your car's brakes operate, this kind of software structure is superior.

LOCAL NETWORKS

Bluetooth does not run TCP/IP, and merging the two is nonsensical. There is a role for local networks, often wireless but sometimes wired, that are not Internet capable and are isolated from the Internet. These networks control local devices. Exposing those local devices to the Internet is an invitation to exploit pacemakers, HVAC control, drone navigation or what have you. This situation is both unlikely to change (requirements do not warrant such networks to connect directly to the Internet) and undesirable (excess complexity without benefit, excess risk and vulnerability).

More capable devices (Class A in this post) are used as gateways between functions using the Internet and data exchanged in the local network. While such devices are dependent on Internet connectivity to function, devices using the local network technology have no such dependency increasing mobility and availability. Class C devices may serve as hubs, or routers in a mobile mesh (see below).

AD HOC MOBILE MESH NETWORKS

There is serious academic, commercial and government interest in inter-networks consisting of mobile devices. These mobile devices route messages, forming an internet larger than any devices wireless range. Classes C, D & E might participate in these, as described.

EXAMPLES

- The cell phone is an example Class A device. Its connection to the Internet may commonly occur via 3G, 4G or LTE standards, or via Wi-Fi / IEEE 802.11 (several variants).
- Your cable TV set top box is another example Class A device, with an extensive user interface and Internet connection via Wi-Fi or various cable Internet standards. I have a DVD player that also qualifies.
- There are refrigerators operating today with Internet connections via 10baseT, 10base100, 10base1000 (gig E) or Wi-Fi. They typically have user interfaces built into the door. This fits the Class B description.
- I have an "atomic clock" with a remote outside temperature sensor. The clock itself meets the description of a minimal Class C device or routine Class B assuming peer to peer coordination, and the sensor a Class E device. I assume the connection is not based on standards, I have not looked in to that.
- I have a speakerphone device in my car that connects to my Android via Bluetooth. The speakerphone device meets the description of a Class D device, and the Android smartphone is again Class A. Bluetooth does not normally use TCP/IP although it can be extended to do so, and Bluetooth Sockets are normally used, an ad-hoc standard.

- Swarming nanobots described in various articles would meet the definition of Class D. They might not use standard network protocols or media due to power and range considerations. Or they might, if somewhat large for nanobots.
- "Smart Dust" sensors would meet the Class E description. They might not use standard Bluetooth either.

INTERNET TO EVERY DEVICE

A contrasting approach envisions every device as connected directly to the Internet, and the Internet accessible to every square inch of the planet. This would require more powerful processors, more powerful transmitters, bigger batteries and greater exposure, with greater cost, to small devices which need none of that. Even if possible due to advances in processors, batteries and wireless networks, such an approach may prove inferior.

4.11 CYBERSPACE



I am a fan of William Gibson. He is the towering master of the cyberpunk genre of science fiction. He once referred to a message I sent him in the foreword of a [book](#), what an honor. He often writes about [cyberspace](#). People would "jack in" a direct neural connection to enter this abstract geospatial realm of cognition. In later works people would wear glasses to perceive [augmented reality](#): the real physical world around us would have additional information and objects superimposed on it.

EARLY VISIONS OF CYBERSPACE HAVE EVOLVED INTO
AUGMENTED REALITY.

Life imitates art. For many years now the US Army has worked to equip soldiers with wearable technologies and reality augmentation. I have referred to this as the most interesting work in the Army. Currently much of this work is conducted at [SPAWAR East](#). I have friends and colleagues down there, working on it today. Augmented reality is coming to the foot soldier.

Of course the USAF has been working on augmented reality for pilots as well, and for a long time. The [F-35 helmet](#) works. Commercial pilots and private pilots are [not far behind](#). Even [drone pilots can now fly with augmented reality](#).

And of course we have augmented reality games and gamification. How long will it be before the user interface of what once was the PC or the smartphone has become an [augmented reality environment](#)? You can point at [Google Glass™](#) as failure, but they were never at the [forefront](#). This stuff is driven by military technology, and is being commercialized. This [product path is not news](#). It was old in WW II.

So start imagining, as you move about during the day, where additional objects and information should be displayed in your field of view. The user interface of today is transient, it is not the end state. The very cutting edge of UI/UX in web design or apps is a primitive thing, and there is more to come.

RESOURCES

VR Gloves:

<https://www.google.com/search?q=vr+gloves&espv=2&biw=1080&bih=1859&tbm=isch&tbo=u&source=univ&sa=X&ved=0CD8QsARqFQoTCMCdj6BgccCFQOggAodGCoGPw#tbm=isch&q=vr+glove>

VR Desktops:

<https://www.google.com/search?q=vr+gloves&espv=2&biw=1080&bih=1859&tbm=isch&tbo=u&source=univ&sa=X&ved=0CD8QsARqFQoTCMCdj6BgccCFQOggAodGCoGPw#tbm=isch&q=VR+desktop>

VR Glasses:

<https://www.google.com/search?q=vr+gloves&espv=2&biw=1080&bih=1859&tbm=isch&tbo=u&source=univ&sa=X&ved=0CD8QsARqFQoTCMCdj6BgccCFQOggAodGCoGPw#tbm=isch&q=VR+glasses>

Augmented Reality Apps:

<https://www.google.com/search?q=vr+gloves&espv=2&biw=1080&bih=1859&tbm=isch&tbo=u&source=univ&sa=>

4.12 A LEGACY SYSTEM UPGRADE, DECEMBER 24, 2014



I once worked on a system ported and updated so many times, over so many years, it was like archaeology. The thing had operated on minicomputers, then it was ported to UNIX, and another variant of UNIX from the first. It was decades old. UNIX was too expensive for the market, so it was ported to Windows.

Furthermore the system needed fault tolerance, and so had been built to access and update two databases simultaneously. If one failed the other would continue. On restoration the databases would be re synchronized. Several hosts running the application could exchange data via the two databases. It was bullet-proof, at the expense of a difficult re-synchronization process including manual steps.

And yet more, if that were not enough, real time asynchronous messages were used between servers for interactive operational coordination, like situation updates. The code layer for coordination and database access was everywhere, underlying everything.

NEED

The company in question wanted to update the system to perform a significant array of new functions not envisioned by the designers. This includes overlay of events on the map, like a geospatial command and control display. This would be overlaid on decades old display software. The customer was unaware that they had just asked the impossible. It was due in a few weeks - short project. Less than 90 days.

Luckily I was working with a small team of all geniuses. Truly, each one a genius or near genius IQ. They found these guys and gathered them together for the task, and I have no idea of how they

found them. These were the most brilliant bunch I had ever worked with, which is saying something unusual. I had worked with the best. I took up leading them and doing architecture.

SCOPE

Foremost, we needed a new map layer. The old code could not be modified without breaking everything, and so we built a whole new layer since the other could not be replaced in the short time-frame. One of the geniuses built a whole new display layer, in different technology, to sit on top of the layers built in the old code. It was a hack, but it was beautiful. A right-click provided the object specific context menu of commands for that sensor or actuator.

Second, we needed something to communicate with and keep track of a wide range of sensors. We built a 3 tier (more on that later) hierarchical real-time (in memory) object database. New object types could be created and inserted. These objects defined and implemented the context menu on the display. The objects also acted as device specific adapters to different sensors and actuators. This monster was my fault, dredged out of old game systems and old sensor system implementations- but another of the geniuses wrote it.

Lastly we needed new forms and functions in the main software. We added a list view and a tree view of the hierarchy, displaying attributes, with an equivalent right click (context) menu. The context menu functions might invoke web forms for data entry, where required. Another great hack, atop all that unworkable legacy code. The third genius wrote that.

SUCCESS

It worked in 90 days. Changes in sensor status caused changes in the displayed sensor icon, in real-time. To silence an alarm or cause an action, just use that right-click context menu. 90 days.

After I left, the boys rewrote it. I had settled on 3 static layers, they rewrote it for n layers. They merged the object DB into the relational DB. I later asked if that had increased sales, or if the additional generality was ever used. It was not. No ROI for the further upgrades, but it paid salary for a time.

We had used scrum, on a whiteboard in my office. It was not truly agile, as the customer was just not involved. The end user would not have understood our hacks to the old legacy code, nor the discussion of them and our judgement calls. In 90 days we had produced a miracle. I suggested it was patent-able but the customer did not appreciate the miracle created.

CONCLUSION

Oh well. This is legacy upgrade, in a microcosm. Customers expect obsolete unsustainable code to be migrated to perform vast new unrelated functions. They think it is routine. What is unusual here is that we had access to the source code. Miracles at a discount and geniuses for hire: What a country.

This particular legacy upgrade was for use as a product. To have a product you need a body of developers capable of maintaining and expanding it. You need enough customers who need the new functions to buy it. It is unwise to pick a narrow niche, get some expensive experts to modify some ancient code-base in ways difficult to maintain, and then have them leave.

4.13 ARCHITECTURE LESSONS OF THE CHAPS FAILURE, OCTOBER 20, 2014

The CHAPS system

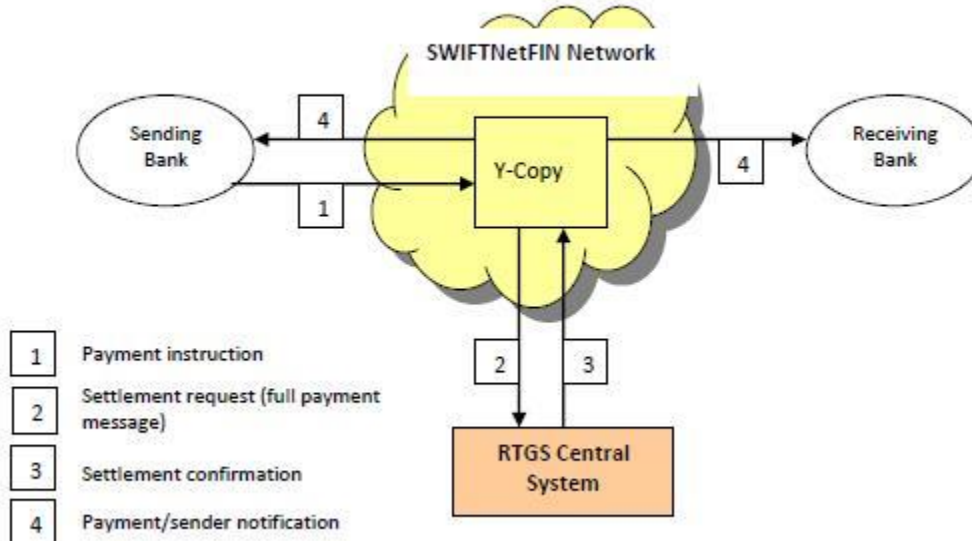


Image from "A Guide to the Bank of England's Real Time Gross Settlement System", Page 6, October 2013. Found at:

<http://www.bankofengland.co.uk/markets/Documents/paymentsystems/rtgsguide.pdf>>

On 20 October 2014 the BOE CHAPS system suffered a 10 hour outage.

WHAT IS CHAPS

"**CHAPS** is the same-day electronic funds transfer system, operated by the bank-owned CHAPS Clearing Company, that is used for high-value/wholesale payments but also for other time-critical lower value payments (such as house purchase). CHAPS payment instructions are routed via the RTGS system and settled individually across the paying and receiving CHAPS banks' settlement accounts. The Bank supports the timely settlement of CHAPS payments through its provision of additional intraday liquidity to the settlement banks; this is provided using eligible securities collateral held in a CHAPS bank's main collateral pool at the Bank."

WHAT IS CREST

"**CREST** is the UK's securities settlement system, operated by Euroclear UK & Ireland, which since November 2001 has provided real-time Delivery versus Payment ultimately against central bank money for transactions in UK securities (gilts, equities and money market instruments). The CREST system settles in a series of very high-frequency cycles through the day; after each cycle the RTGS system is advised of the debits and credits to be made to the CREST settlement banks' accounts as a result of the settlement activity performed by CREST in that cycle. The Bank supports the real-time

settlement process in CREST through the provision of intraday liquidity to the CREST settlement banks; and this is provided via a same-day repo (under a procedure known as auto-collateralisation)."

<http://www.bankofengland.co.uk/markets/Pages/paymentsystems/default.aspx>

Both CHAPS and CREST are centralized individual systems hanging off the SWIFT network. They are connected to, or perhaps part of the RTGS system environment. As we have just seen, the health of a big chunk of the western world's economy depends on the operation of CHAPS. So also CREST.

SWIFT

SWIFT is a network for financial transactions designed as a fault tolerant mesh, where if any part fails, the rest keeps moving. It is also a set of messaging protocols designed to facilitate that. SWIFT is designed to have no single point of failure.

SINGLE POINTS OF FAILURE

To build a reliable large system you avoid single points of failure. These are individual components that can cause failure of the entire larger system. Large centralized subsystems like RTGS, CHAPS and CREST are single points of failure.

https://en.wikipedia.org/wiki/Single_point_of_failure

DISTRIBUTED SYSTEMS

We could envision CHAPS or CREST as distributed systems without single points of failure. Each bank would handle its part of the transaction without a central broker. This might be aided by many redundant routing nodes in a mesh, using asynchronous messaging standards (cf. SWIFT system)

<http://www0.cs.ucl.ac.uk/staff/ucacwxe/lectures/ds98-99/dsee3.pdf>

<http://www.ebizq.net/topics/standards/features/1638.html>

HALFWAY THERE

You can partially mitigate the problem(s) (of single points of failure) by creating hot spare sites, warm spare sites, cold spare sites for COOP (Continuity of Operations). These are operated at alternate sites, safe from danger. As we have seen here, there are limits to the effectiveness of that approach, and the time to "cut-over" to the backup system. If you put the backup site far enough away, it is too far to update in real-time.

<http://www.ciupdate.com/trends/article.php/3872926/Disaster-Recovery-Planning---How-Far-is-Far-Enough.htm>

RESILIENCE

Building a world resistant to hacking, terrorists, natural disasters, administrative foul-ups and such depends on avoiding centralized brokers (and huge central SOA systems). For those systems critical

to the public good and the continuity of society, you need to put some extra effort into design to cause them to be fault tolerant and to fail gracefully. When one part fails the rest should continue operations. Otherwise one nut with a bomb might destroy civilization.

<http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5361311&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F5357317%2F5361202%2F05361311.pdf%3Farnumber%3D5361311>

CONCLUSION

Build in redundancy, fault tolerance and distributed systems concepts for large mission critical systems.

4.14 DEFEATING DDOS, DECEMBER 24, 2014



Here is a story seldom told. By 2001, several of us had formed a company and had built the Arcturus(tm) product line. These products were designed to archive, organize and protect documents. There were several products:

- A reusable software component for creating new workflow applications
- A desktop application for routing storage and retrieval of documents
- An administrative application with a workflow editor
- A secure Web Server for extranet access of documents

We had lost the window to obtain venture capital in the Dot Com bust. Our infrastructure was modest. We had four servers in my basement closet connected to a business internet modem and my house network. Development was usually remote.

THE ENEMY

One morning we noticed that our site had been defaced. Apparently some Chinese hackers had mistaken our small development effort for a government project.

This was annoying, our web product was intended to be secure. Instead of keeping all files on the server, it fetched, decompressed and decrypted them as needed. It kept only cached copies, and a few main site pages.

We posted a note to the hackers on our new homepage that this was a small company, and not a government project. We went back to work to improve security by moving those few pages out, to be decrypted and fetched on the fly like the other content. Having done so, we then experienced some various attacks on bits of our Windows™ servers that were not "locked down". We secured those.

LEMONS TO LEMONADE

Then, as we could not convince them to go away, we invited the Chinese hackers to try to crack the new version. The Homepage was now protected, and included a challenge.

At this point our story begins in earnest. Thwarted, our black-hat hackers began other attack types, and we tightened up the firewall. We locked down every unneeded incoming service port. They responded with a DDOS attack.

For those not familiar with DDOS, many (dozens to hundreds) of other people's servers are hacked and commandeered. These servers belonging to others are enlisted to attack a target, or a few targets, with some range of malicious exploits. The sheer volume may shut you down, even if the exploits are rejected by your infrastructure.

Every jerk with a nonsensical idea had made millions, and we were locked in battle with state sponsored hackers. This made us life-long geeks angry. The lead programmer had written his own compiler just to learn C many years ago. I was the architect. There was another R&D engineer and an expert Web designer in the mix, as well. The accountant was helping us out from a fortune 50 company down the road, part time, for equity. We were not easily intimidated.

THE NEXT LEVEL

How would we defeat DDOS? We detected attack patterns and sources and rejected them, answering with only silence. This was 2001, if you will remember, and the industry had not yet inserted these kinds of features into firewalls.

We tested the results. Arcturus™ continued to run just fine while under attack so great we had to increase our internet bandwidth. We posted a new homepage, with a Chinese character meaning "skill" and a message saying essentially that "our Kung Fu is better than yours".

Days passed. The Arcturus™ Web Server continued to operate. No problem. It was a bit slow due to the massive attack traffic on the WAN, not too bad.

A CHERRY ON TOP

Then they hacked the cable router, opened a port, and executed what we believe to be the first direct attack against a SQL Server RPC connection. Ouch. No problem, just one more tweak to us Alpha-Geeks. We moved our SQL Servers to XNS, and removed the TCP/IP protocol entirely. Poof!

Ineffective DDOS and SQL attacks occurred for weeks, with no success. We added some verbiage to our homepage, taunting them. No further holes were found in the Web server. They did hack our externally hosted email accounts, but Arcturus™ had an internal mail mechanism that remained secure and we told them so.

We had won. Our Fu was supreme. In those days hacking battles between experts were showdowns, we talked like that. We were done. It was over for us.

CONCLUSION

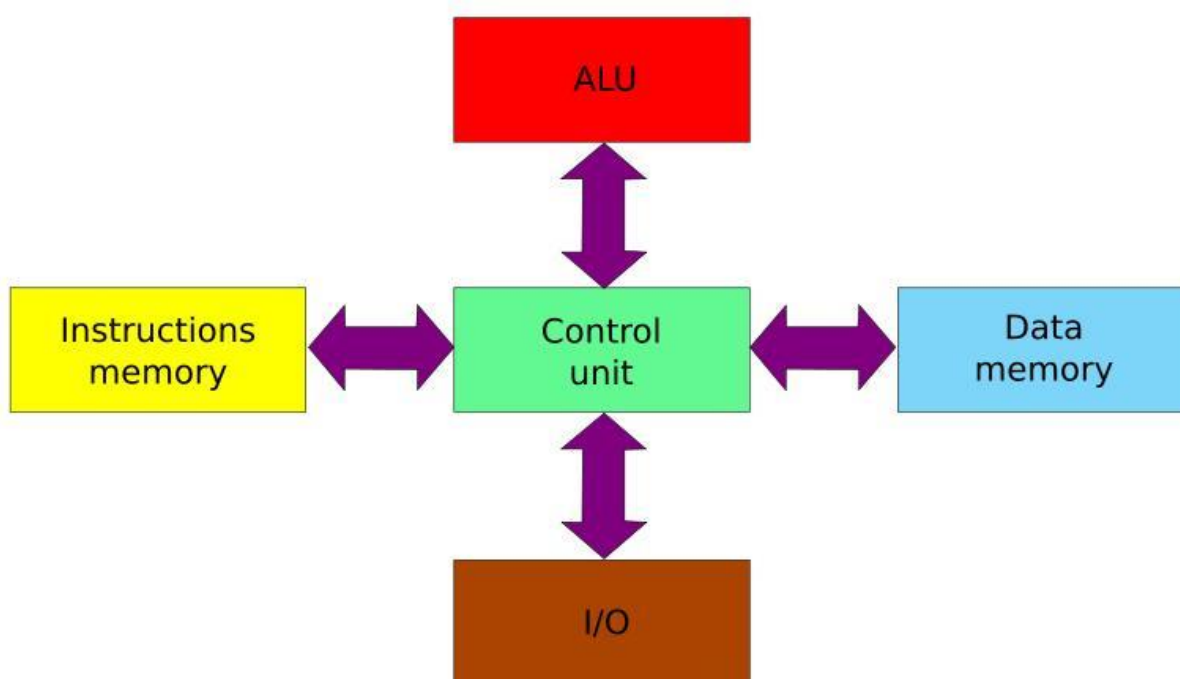
So why are your Web servers still being hacked daily? Well, we were out of money. We had no VC. I was broke, my wife was mad, and we folded up the whole thing after a few years of no further action.

What, you still need this today? Web servers are still vulnerable to DDOS right now? OK, send me a message. Bring cash. I am done financing this from my own pocket. Recently I had discussed adding special purpose hardware to eliminate the few remaining exploits in a Web Server, and that would take a few bucks. It is all easily possible, and we have the technology for most of it today, and a path forward for R&D.

Now if I can only find my old white hat...

"hmm hmm hm hmm-hmm... gimme back my bullets.."

4.15 ADVANCED SECURITY FROM THE TRASH BIN, OCTOBER 15, 2014



Possibly the most effective means to combat IT security threats can be found in the vast trash bin of old technology. I will describe five technologies that could completely redraw the cybersecurity landscape overnight.

HARVARD RISC ARCHITECTURE

Back in the 1980s and 1990s RISC was a big research topic. Harvard had a novel approach to pushing more throughput through the processor, using different data and instruction buses. This separate instruction bus and the separate instruction memory that went with it are the technology in question. A distinct, physically or virtually isolated disk drive might contain the executable code, unmixed with data. In this type of CPU programs downloaded from the Web to data memory would be physically incapable of executing. Read that again if you missed it.

(We do not need the reduced instruction set really, just the two bus architecture. You could build an i86 like processor with 2 buses, a few instructions would perhaps execute differently.)

Interpreted scripts would still be a potential problem, as the script could be downloaded and executed from data memory, but the interpreter would be loaded in instruction memory. Any security in the interpreter could not be bypassed by code modification. This was all discussed at the time, old prior art.

https://en.wikipedia.org/wiki/Harvard_architecture

MAINTENANCE PROCESSORS

Common in the old days of computing, and perhaps most featured in the IBM 4341 mainframe, maintenance processors would manage processes like patch updates and troubleshooting. Imagine a separate processor, with OS and code not accessible by the Internet, fetching and applying only authorized verified patches. Back to the future, or forward to the past.

https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP4341.html

OTHER PROTOCOL STACKS

When we wanted to isolate a database server, say, from the Internet we used to sometimes run only XNS on it. Imagine your internal browsers, all of them, running XNS only to safe servers. Alternately imagine the Intranet maintenance bus for patches running XNS only. Halt the routing of XNS at the boundary. Spoof that.

<http://www.protocols.com/pbook/xns.htm>

MMU AND SAFE OPERATING SYSTEMS

Not that long ago operating systems could isolate one process, one job, one task so that it was physically impossible to access or infect another task or the OS itself. Today we use VM software that may isolate one VM running one OS from another, but the OS is still vulnerable inside that VM. The older way was perhaps a bit slower on a real machine, but safer. Now we incur some of the old overhead anyway in the VM software. Reexamination seems reasonable.

https://en.wikipedia.org/wiki/Memory_management_unit

READ ONLY MEMORY AND DISKS

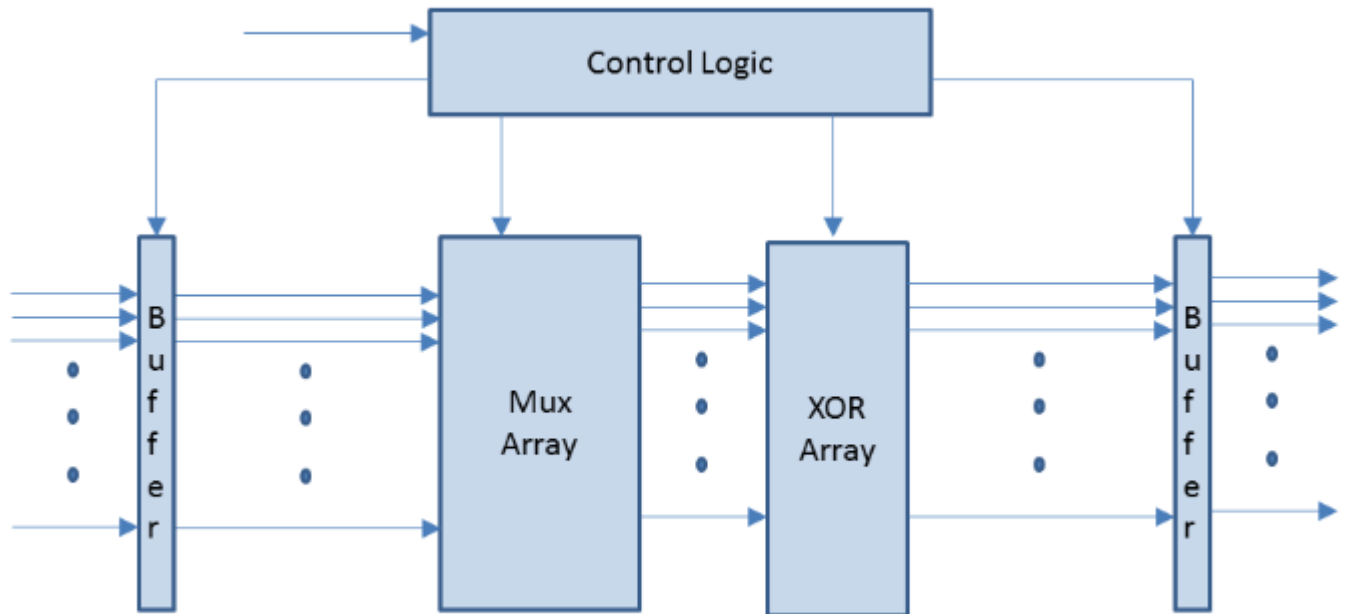
Once upon a time if you wanted new software on your embedded device you needed a new chip, new physical media, or a physical key, to load it. No excuses. It was harder, but safe.

https://en.wikipedia.org/wiki/Read-only_memory

CONCLUSIONS

There are no new technologies or ideas in this post, just old ideas that worked. We are losing billions to cybercrime and attacks right now. We do not need to. These unused old technologies could change the whole picture. A line of safe servers designed to repel that crime, those attacks is possible. Do we have the will to transcend this problem?

4.15 CRYPTO AND THE MULTIPLEXER ARRAY, JUNE 17, 2015



So you are a security architect? Here is something that you may have never heard of. There is a very efficient means to encrypt and decrypt that is not commonly used, and it provides lightning fast hardware acceleration.

Imagine you have a set of 16 marbles before you, some white and some black. They are in little plastic cups labeled 1 to 16. Now imagine a second set of cups, also labeled 1 to 16, but out of order. Take the labels off. Imagine a device that can, in a single motion, move all your marbles from the ordered cups to the scrambled cups and put each in the right place.

Now without the map from the new order to the original it will be difficult for someone to put the marbles back in order, and see the original sequence of blacks and white marbles. However our device can put them back in order just as quickly.

Now if we talk bits and not marbles, and thousands of bits not just 16, you have a simple machine for encryption by reordering bits. On analysis, the machine is not all that good, because you can still count the blacks and whites- but a small addition to flip from black to white (one to zero) and back per some pattern fixes that.

The hardware which does the basic operation is called a multiplexer, selecting one input to replicate that value on the output. You can get it in an old TTL chip or in a custom chip function library or replicate it in an FPGA chip or ASIC. It is basic digital circuitry. (A variation would have an XOR array both before and after the multiplexor array.)

To do it to many bits at once you need an array of multiplexers. For 16 bits, there will be 16 multiplexers. To add the bit-flipping you need an XOR gate array on the end. As digital electronics go, very simple stuff. You need some other logic to coordinate all the multiplexers and XOR gates and buffer the bits, still simple.

However with this kind of hardware you can obscure blocks of bits in tiny fractions of a second that will take a conventional processor much longer to undo. Perhaps, depending on the number of bits, it might be many thousands of times longer. Hook this up to an algorithm to change how the bits are swapped and changed and you have a powerful block encryption/decryption co-processor.

If the information to reorder is M , and the information to flip some bits (and not others) is N , then each block should have a different M and N , (You want " M " to only produce valid distinct sequences of the bits, no redundancy and no dropping bits.) The algorithm changes these for each block encrypted (and later decrypted). There is no particular reason why a block cannot be 1K bytes or more, if you have the chip size for it.

The size of an encrypted block is dependent on the hardware, and the hardware cannot swap or flip bits outside its block size. You could overlap blocks however, and thereby swap bits between blocks. Ultimately the strength of the encryption produced would be a function of the control algorithm: If you used the same M and N for every block it would be very weak, but if M and N were taken from a ["1 time pad"](#) the encryption provided might be un-crack-able.

This is as true today as when I first looked at it in college, 30 years ago. However it has not been exploited much. It has the advantage of avoiding all those hard operations, like multiplication, that slow down crypto.

4.17 A MISSING UNIX NETWORK SERVICE, JUNE 17, 2015



It should be there. Anyone who studies the UNIX network services can tell you it should be there. (I'll just say UNIX for brevity and imply Linux as well so you can flame me later.) It was overtaken by events, and thwarted by export laws, even prevented by politics - but it should be there. It kind of bothers me, this absence. It is so darned obvious.

For those unaware UNIX has [network services](#). They are assigned to port numbers, like phone extensions that can get you to that particular final party in your conversation. These have been around for some time, nearly forever in UNIX-dog years. They are a comprehensive set nifty tools for doing things across a network. They include (just a few):

- "sysstat"
- FTP
- telnet
- SMTP
- HTTP
- DNS... and generally everything that makes the Internet run.

But there is no simple secure symmetric encryption interconnection service. It is missing. Just missing... I find it odd. So what would this missing thing be like?

PURPOSE

The missing service would connect you from one UNIX system to another, using efficient and powerful point-to-point symmetric encryption, without resorting to external authorities for trust

(like PKI), and without resorting to complex and inefficient asymmetric encryption. Probably it would result in an encrypted telnet session, or maybe an HTTP session. Maybe a generic stream...

CONFIGURATION FILE

Like most UNIX services, an ordinary configuration file in /etc. or some other well-known location would list the partner machines that this one can communicate with by our service, and the keys for communicating to those machines, one by one. A date of last key update would be included. The encryption algorithm to be used with that partner might also be listed, line by line, partner-by-partner - or it could be dynamically set in session initiation. Fancier mechanisms involving storing the keys as encrypted might occur as well, and the rest of normal precaution. No configuration entry, no "connectee"... or something like that.

CARE AND FEEDING

When the administrator wants to establish trust between his machine and a distant machine, he calls the administrator of the other machine. Over the phone (out of band) they agree on an initial key, and flag the line in the configuration file as initial.

For initial connections and old keys the one server will connect with the other and perform a Diffie-Hellman (D-H) protocol coordination to set a brand new key. The D-H exchange would be encrypted using the old key, perhaps, to thwart man-in-the-middle attacks. This key will be used until one machine or another decides to connect up and change it. Thereafter, until again changed, the service will send and receive encrypted data for telnet (or maybe HTTP) sessions.

EXAMPLE:

Ring...

Bob: This is Bob.

Alice: Hi Bob, lets configure S3E between my server Aardvark and your server Bulldog, as we discussed.

Bob: OK, let's use Psalm 23 from that weird bible version I sent you as the initial key. Start copying at "Though I walk through the dark alley, I will fear no muggers" and go to the end of the page there. This version of S3E accepts ASCII initial keys.

Alice: OK.

Time passes...

Alice: Got it.

Bob: Me too. I will init S3E to contact your Aardvark server and produce real keys. Here we go.

Time passes...

Alice: I have 3E FF 23 96 54 etc.

Bob: Me too. We have success. See you later Alice.

Click.

ADVANTAGES

Using symmetric encryption is efficient. Large keys can and are commonly used, and a good implementation could resist even state sponsored attacks using supercomputers. Many symmetric algorithms are highly resistant to cracking, and probably will remain so for a very long time. You need not worry about the root certificate being compromised, or expiring, as there is none. In short, everything about hacking encrypted communications would change.

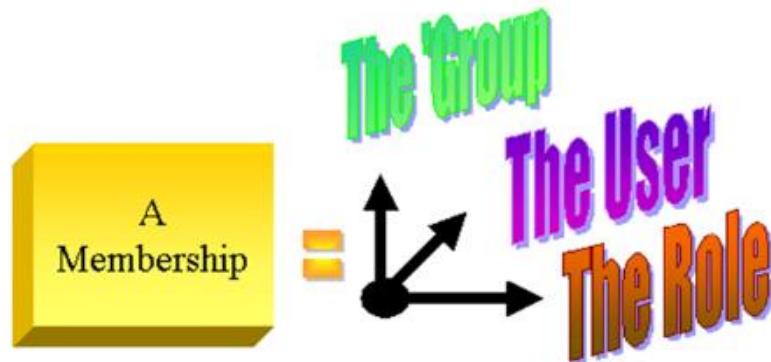
The main difference between the conjectured service and commonly used mechanisms like SSL or SSH lies in the chain of trust. In the service described only "private keys" aka symmetric keys are used. In SSL or SSH "public keys" are also used, and you rely on a third party to establish trust between your system and another. However public keys have been [widely compromised lately](#). In the described service one administrator directly knows and trusts another, without recourse to any third party. While you cannot encompass the world with such a system, it is more secure for groups of servers in a limited community.

But the conjectured service was (as far as I know) never built, never distributed and never commonly used. I find it odd. It seems quite strange. It still has very broad applicability, this missing service, in light of the cracked and hacked root certificates and such. Maybe we are missing an important or even critical tool in the toolbox.

Maybe somebody will write it before I do. Just Sayin'...

4.18 ANOTHER ACCESS CONTROL MODEL, JUNE 6, 2015

Security Subsystem:



A Membership has Three Dimensions!

In security the access control model is a central topic. The Microsoft Windows™ operating system has an access control model built in. Unix/Linux also has an access control model built in. Databases such as Oracle™ and MS SQL Server™ also have built in access control models. These are all studied heavily in security education and training, and in certifications such as the CISSP™ of (ISC)2.

However, there are other access control models. I want to briefly write about a different model, suitable to application level security. This model was used in the Arcturus™ product line, mentioned earlier (a former product at a past company that missed the VC window of opportunity in the Dot Com boom). This access control model is useful, was implemented, and works great.

Application level security is that security used to control access enterprise application features, common in enterprise software accessing database screens. The mechanism is simple, so this will be a short post. Do not let the simplicity confuse you as to the power of the access control model.

GROUP

In this model access is managed by the concept of a membership in a workgroup or group. The group may have workflows associated with it, or other flow specific resources such as screens or web pages. Files or datasets may also be available to that group. The point is to moderate the resources of the group by means of the membership.

USER

The user is a user account, an identity. One user should have only one account. The user construct uniquely identifies an individual.

A user will also have a "level of trust", analogous to a government security clearance, indicating to what degree they will be trusted.

ROLE

The role construct describe the function or work which a user performs. By virtue of a role the user may have access to the resources of the group.

MEMBERSHIP

The membership is the intersection, the combination of a user performing a role in a group. 'Simple.

PERMISSIONS

Permissions may be assigned to a user, a role or a group to access some object.

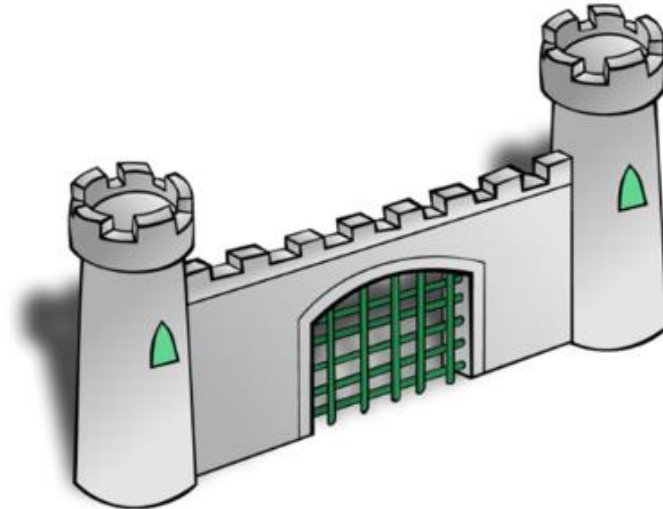
- In a role based permission you also check if the user is a member of the group that owns the resource. You also check the level of trust. Proper implementation allows similar resources for the same role to be accessed in different groups, whatever group is accessed, for users with the same role in multiple groups. For example group administrator screens which are the same for all groups have the same permission names and types.
- In a user based permission, you check to see of the level of trust is adequate to access the resource.
- In a group based permission you check if the user's level of trust is adequate to access the resource.

ADVANTAGES

This type of access control⁵ implemented in application security systems better models the compartmentalized security used in most highly secure organizations. Further, when the user changes membership from one group to another, the permissions change to fit the role in that group - as appropriate. The access control model is simple to implement, lightweight and effective. It is a natural fit for implementation in a relational database. It's not earthshaking or particularly innovative, but it is simple and effective.

⁵ This is closer to the control mechanism used on System 38, AS400, Systemi machines which had/have the best features around work, access etc. so innovative for its time, years ahead. I spent a lot of my time in IBM midrange rather than Unix and Windows. Paul vereycken

4.19 SECURITY CHALLENGE-RESPONSE EXAMPLE, JUNE 25, 2015



In security there is a type of protocol referred to as challenge-response. In old movies you will see a posted guard ask for the "word of the day" in the dark to know if shooting or admitting the visitor is most appropriate. In spy movies you may see one spy state a phrase, and the other respond with an unlikely phrase to verify to both that they have the correct connection although they had never met before. Passwords are said to be a simple type of challenge response, as are RSA "tokens" which give you a unique number based on the serial number of the device and the time.

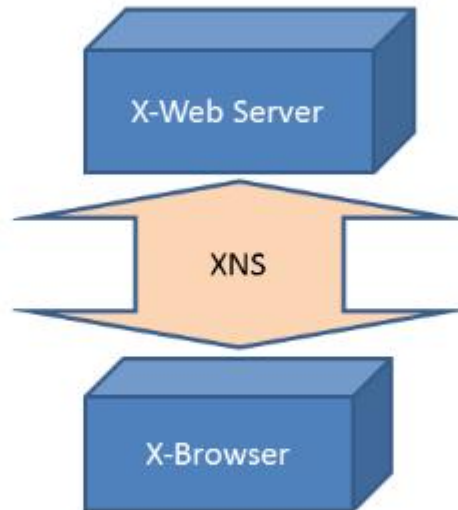
But suppose you have an information system to secure, and desire some authentication method as strong as RSA but without hardware. How would you do that using a challenge-response type protocol? Here is a simple method to mull over.

For your users print a response booklet which is valid for only some period. In the book challenge words are on the left, response words are on the right. At login, have the system present one of the challenge words. The user must type in the unique correct response word. 'Simple.

Of course you must print and distribute new booklets often, as the pairs may be rapidly compromised when used. You might use one pair only once ever. You must take care to distribute booklets securely. You might perhaps distribute different booklets to different account holders. Booklets can have different columns of response, and only the user knows which column to use when.

There you have it: no gadgets and simple to implement. It can combine something you know with something you have. Go forth and do good things!

4.20 HOW TO DO ARCHITECTURE: X-BROWSER, JULY 2, 2015



This is an example of architecture, applied to security. There are three main platform components that cause the majority of vulnerability exposures in the enterprise. This is not news to practitioners. These are email, the web browser, and the web server. What if we fundamentally changed the way this technology is configured? Could we reduce the exposed footprint to be exploited, without too much expense?

We will take some existing technologies and rearrange the relationships between them, providing some new operating principles, to create a new architecture. The components themselves will be little changed. Because new technologies will not be developed, the proposed architecture should be inexpensive to implement. The result should be an example of thinking like an architect. The technologies we will refer to will be the browser, the web server, email, the old XNS protocol, and virtualization. We will also mention FTP.

Now here is the use case, for you who are into use cases. You have a corporate/enterprise user, at home or at work, and he needs to run the enterprise applications securely, access enterprise resources securely, free of various common threats in today's Internet. Call it an epic, if you like. To follow along you might stop thinking about what code to write, as that will not be the main point.

So let us first rethink some old notions. We will put Joe User in a different, secure virtual machine on his PC. He has an insecure standard VM, but this is different. It contains secure approved tested versions of applications only, and he cannot install new insecure software here. This VM cannot access the Internet, and no TCP/IP is configured here. Joe and the enterprise are safe here.

Now let's conjecture a company or enterprise intranet. It is not the Internet, and is distinct from it. But rather than running new wires or whatever, we will simply use a different protocol, so the intranet traffic can run side by side next to the vulnerable TCP/IP traffic. To avoid any new, expensive development we will use plain old XNS. XNS is a network protocol stack developed in the early days by Xerox™ and used by Netware™ and Banyan™ and others for many years. It is proven and tested, and once supported massive networks like the USMC Intranet.

Now let's take the open source browser code from your favorite supplier, and bind it to use only XNS. It will no longer be capable of using TCP/IP at all. Otherwise all the features will be present. (Some coding is implied here, but it is not the point, so we will move on.) We will call it the X-Browser.

To serve Web pages to our browser we will need a web server. We will take an open source Web server and cause it to use only XNS, and TCP/IP will not be available to it. Otherwise all features will be present. (Again some coding is implied here, not the point.) We will call it the X-Server.

Now our corporation can use the X-Server and X-Browser for all Web enabled enterprise applications without fear of the vast majority of exploits on the Web. But function is limited by this choice, so let's make the X-browser and X-Server more capable with some other features.

- If an external TCP/IP link is accessed in the browser, the X-Server will act as a proxy to retrieve the page, and will clean it thoroughly to remove threats before passing it on. Worms, cross site scripting, viruses, keystroke loggers will all be removed and the content blocked. It will not be possible for the user to decrease the filtering, and some function will be lost in the secure intranet to preserve security.
- While browsers currently only handle HTTP, the X-Browser (and designated/configured X-Servers) will also handle enterprise email and FTP in the same tool. This will simplify distribution and support. Let's assume drag and drop functionality for all this, a slick implementation. All will use only XNS between X-Browser and X-Server. Security filtering will also be applied here, without exception.

Will this remove all vulnerabilities? No, not all. Disconnecting the X-Server from TCP/IP except in specific controlled instances will help. Altogether, the approach should eliminate the majority of routine enterprise vulnerabilities, won't it? Here is a new architecture, an example, constructed of mainly existing technology. The job is now to analyze and see if the proposed architecture meets the needs. What did we leave out? What could be removed? Is it better that way?

Once we have figured that out, separately, we can predict the ROI. Then we can compare the notion of transforming IT and the enterprise using this approach, comparing it to other proposed operational changes, and see if this initiative should be funded. The former, above, is solution architecture and the latter, in this paragraph, is enterprise architecture.

What do you think: Is this a worthy effort? Why or why not? How would you compare it to the value of other efforts? Will the costs be high or low? Will the returns be high or low? If this were a product line, could it be used by the small enterprise as well as the large enterprise (does it scale well)?

You can share XNS with partners. Suppose all your partner companies share the XNS intranet. All B2B traffic happens over the intranet. How does that effect ROI and security? What if there was a

national intranet? How would that affect APT (Advanced Persistent Threats) and state sponsored intellectual property theft?

ENDING REMARKS FOR VOLUME 3

This concludes Volume 3, Solution Architecture. Having read it, the nature of solution level versus enterprise level architecture or segment level architecture should be clear. At the lowest level of the hierarchy solution level architecture is most detailed, and of the most narrow scope.

We have discussed the totality of the enterprise architecture space in Section 1. We have examined the nature of solution level architecture in Section 2. We have discussed some selected topics of interest in Section 3. Finally we have seen examples of solutions and parts of solutions in Section 4. This has been a survey, scratching the surface of many areas. Its purpose is to give you overall perspective as an architect, not to educate you in the deepest details of any one issue.

Assuming this volume has been of use to you,, you might want to read more. To understand enterprise level or solution level architecture, you would now read volumes 1 and 2. To understand governance and maturity management in enterprise architecture you would read Volume 4. To understand the business context, and all the vexing details of the environment in which we try to produce working architecture, you would read Volume 5.

For depth instead of breadth there are many textbooks on solution architecture and systems engineering. This volume is not intended to supplant them. You cannot read them all, but to be any good at this you should read several.

Good luck in your journey. I hope my eBook has been of use.