What's in the Code
Unraveling the Enigma of Legacy Systems

l**ogical** s**olutions©**

adjusted to the need

Wouldn't it be brilliant to have a logical solution to:

IDENTIFY SERVICES:

- Start with your business analysis, identify services from existing work flows and identify new services not currently provided.
- For the existing services, identify online queries/data entry and batch jobs
  - identify the programs associated with each online query/data entry
  - identify the business rules as compared to what is running in production
  - identify the data files and database tables associated with each online query/data entry
  - identify the data fields associated with each online query/data entry
  - identify the programs, data files and database tables, and data fields associated with each batch job
  - identify all other programs that also access or update those data files/tables
- For new services
  - identify the data files and database tables required for the service
  - identify the programs that utilize those files and database tables
  - identify the data fields used and updated by those programs
  - identify all other programs that also access or update those data files/tables

System/programming languages have over time evolved from the 40'ies when the first "modern" computers were developer and because of the limited speed, memory, and capabilities, programming was accomplished using low level machine programming language such as assembler.

Today, most legacy system are written in COBOL, ASSEMBLER, PL/1 and REXX

And the programming evolution continues today. The industry also saw a shift from procedural languages such as COBOL to object orientated languages such as JAVA

CONCEPT:

SEPARATION OF CONCERNS

Multi-dimensional separation of concerns is an approach to separation of concerns, supporting construction, evolution and integration of software. Its goals are to enable:

Encapsulation of all kinds of concerns in a software system, simultaneously.
Overlapping and interacting concerns.
On-demand re-modularization.

Separation of concerns is a concept that is at the core of software engineering. It refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concern (concept, goal, purpose, etc.). Concerns are the primary motivation for organizing and decomposing software into manageable and comprehensible parts. Many kinds of concerns may be relevant to different developers in different roles, or at different stages of the software lifecycle. Appropriate separation of concerns has been hypothesized to reduce software complexity and improve comprehensibility; promote traceability; facilitate reuse, non-invasive adaptation, customization, and evolution; and simplify component integration.

The term multi-dimensional separation of concerns (MDSOC) refers to flexible and incremental separation, modularization, and integration of software artifacts based on any number of concerns. It overcomes limitations of existing mechanisms by permitting clean separation of multiple, potentially overlapping and interacting concerns simultaneously. MDSOC promotes reuse, improves comprehension, reduces the impact of change, eases maintenance and evolution, improves traceability, and opens the door to system refactoring and reengineering.

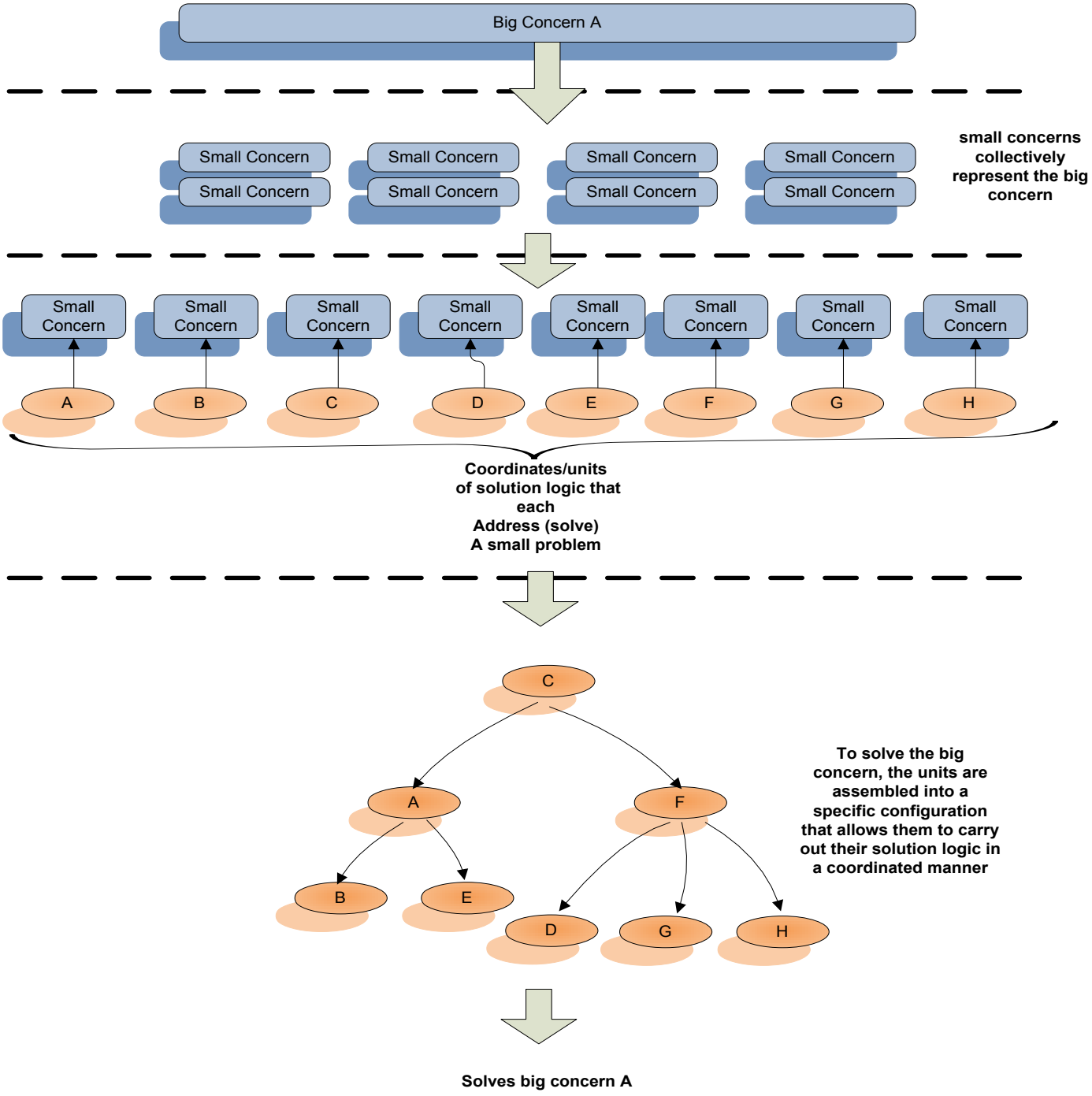
The separation allows:

- To allow people to work on individual pieces of the system in isolation;
- To facilitate reusability;
- To ensure the maintainability of a system;
- To add new features easily;
- To enable everyone to better understand the system;
- To allow support for multi-dimensional separation of concerns.

Remember, a dimension of concern is simply an approach to decomposing, organizing, and structuring software according to concerns of a particular kind. It would be brilliant to have a logical solution, wouldn’t it?

CONCEPTUAL VISUALIZATION:

SEPARATION OF CONCERNS



The separation of concerns theory encourages us to break down a larger problem into multiple smaller problems (concerns). This gives us the opportunity to build corresponding pieces of solution logic, each of which solves a small problem (address an individual concern). These capabilities are part of units that are assembled into a composition through which they are coordinated to collectively solve the large problem.

A logical solution should be able to be used with the languages shown in the table below. If one of the objectives is to convert for the legacy code to one of these other languages – how do you research and document what's in the legacy code. The same question is relevant for those needing to change, maintain or even evolve from the legacy code.

Supported	Code		Flow Chart Capable
ASSEMBLER	**YES**		NO
BIND	**YES**		NO
BMS	**YES**		NO
C / C++	**YES**		YES
CICS	YES		YES
COBOL	**YES**		YES
DB2 PREDICATE	**YES**		YES
DB2 TABLE	**YES**		YES
DDL	**YES**		NO
FORTRAN	**YES**		YES
IDMS	**YES**		YES
IMS	YES		YES
JAVA	**YES**		YES
PASCAL	**YES**		YES
PL/1	**YES**		YES
POWERBUILDER	**YES**	*	YES
REXX	**YES**		YES
RPG	**YES**		NO
SOA	**YES**		YES
SORT	**YES**		NO
SYS1.PARMLIB	**YES**		NO
VSAM (LISTCAT)	**YES**		YES
XML	**YES**		NO
z/OS JCL	**YES**		YES
*	upload to mainframe		

With this evolution it's important to understand that system are still dependent on code developed in the past.

The deeper you go the more good things you learn about in your Legacy Systems an industry that supports 240 billion lines of Cobol code running active business applications worldwide.

Legacy systems powered the past but they can power the future. They have been tuned and streamlined over many years, and they most likely perform very well. Leveraging these existing assets in place, where possible, is the best approach to capitalizing on the investment and years of work spent perfecting them.

These systems could be the building blocks that fuel your future. So, as you discover more good things in these systems think about how they can help you reclaim leadership and leverage when it comes to transforming your business to a new model.

With so many lines of code in production systems how is it that someone can navigate to what the concern is or what is important?

UNRAVELING THE ENIGMA OF LEGACY SYSTEMS

DO YOU THINK YOU KNOW YOUR LEGACY SYSTEMS?

Do you have a technology to effectively pass your mainframe legacy systems down through time? Legacy systems have been tuned and streamlined over many years, and they most likely perform very well. Leveraging these existing assets in place, where possible, is the best approach to capitalize on your investment and years of work spent perfecting them.

A logical solution can resolve many of the issues surrounding legacy systems especially the issue of retiring baby boomers as it relates to passing the knowledge onto a new generation.

If your systems were developed in the ‘70s or 80’s, it will be tougher and tougher to find people with the skills and desire to work on them. Waiting to modernize will only cost you much more later. If you think programmers are having a hard time determining what an old COBOL program is doing, consider the difficulty in deciphering hundreds or even thousands of new interfaces that were built on top of legacy architecture. The business rules from yesteryear are probably not appropriate for todays’ world.
A logical course of action would be to suggest knowledge transfer, but that will not completely resolve the situation because of two significant issues.

The first is the applications that were installed on your mainframe decades ago were consistently tweaked, updated, and enhanced through the years.

It would be impossible to review every change made along the way to pick up exactly where a COBOL expert with 30 years of experience left off.

This leads to the second issue—experience. It is simply not possible to transfer this knowledge from the retiring workforce and expect a new team of developers to be as conversant in COBOL as people who have dedicated their careers to it.

THE LOGICAL SOLUTION FITS INTO THREE MAJOR CATEGORIES:

*ROOT CAUSE ANALYSIS
*PLANNING VISIBILITY
*THE ABILITY TO CREATE AND DRIVE TRANSFORMATION ROADMAPS

However, when the knowledge base leaves, who will maintain and continue development of these systems?

How will they navigate this vast environment to find and address the areas of concern?

Who is training this next generation?

What tools will you have?

How will impact analysis be performed?

The logical solution being proposed can:

1. Assess the Impact of Change.

   - Know the full impact for the project before starting a project.
   - Formulate a cohesive blueprint of the enterprise.
   - Make business decisions quickly and with resolve.
   - Prevent teams from going off on parallel and conflicting paths.
   - Allow developers see outside the box instead of having a limited sphere of vision.
   - Provide cross-disciplinary, cross-functional transparency and allow developers to rapidly visualize interdependent components working on related projects.
   - Prevent stymied efforts to craft and deploy efficient, cost effective solutions.
   - Allow the development teams to zoom in on progressively granular views of issues and possible solutions without being blurred or off target.

2. Reliability and Maintainability.

   - Know the number of modifications, where located and complexity involved.

3. Job Satisfaction.

   - Developer will feel in control by having a map and reliable information that guides their effort.

4. Productivity.

"Know what you're taking on before you're in the middle of an effort and find significant unknowns."

- Simultaneously search up to 500+ concerns of interest.
- Minimum learning curve
- Each developer will be able to work and address modification more effectively.
- Estimate Project and work assignments correctly.
- Determine the full project impact and scope before starting.

5. Business Logic.

- Highlights and identifies business logic implemented in the code.
- Code can be extracted down to PC platform.
- Code can be flowcharted individually or batched via our 3rd party partner software.
- Complexity of the code determined and evaluated via our 3rd party partner software.
- Allows for the editing of business logic graphically in flowchart mode via our 3rd party partner software.

6. Regulatory Requirement Support.

- Document the implemented business logic as opposed to the business rules believed to be in place.
- Know what is running in production as opposed to what you think is running in production.

RIPPLE-TRAC is named for the Ripple effect -a series of repercussions or consequences of change particularly appropriate to the software industry where the effects of one event set off other unexpected events.

The technology engaged by RIPPLE-TRAC presents the information from the point of view of the concern – the architect only sees information that is related to the concern and has control on what concerns should be brought to the forefront or obscured into the background. The uniqueness of RIPPLE-TRAC is the targeted approach, which we believe has potential to support longer term refactoring of application logic into reusable components and services (SOA). RIPPLE-TRAC can also be used as a support tool for migrating from one platform to another.

RIPPLE-TRAC is a technology for seeing wholes. It is a framework for seeing inter-relationships rather than things, for seeing patterns for change rather than static "snapshots." Legacy complexity can easily undermine confidence and responsibility. RIPPLE-TRAC is the antidote to this sense of helplessness that many feel as we try to understand legacy systems. RIPPLE-TRAC is a technology for seeing the "structures"

that underlie complex relationships, and for discerning high from low leverage change. By seeing wholes we learn how to restructure a relationship.

The RIPPLE-TRAC solution can help your team test software more thoroughly and more quickly. Manually analyzing your software applications can result in late releases or inconsistent results. Ad hoc processes for managing your analysis efforts can't ensure the kind of quality you need. By automating your more labor-intensive analysis tasks with RIPPLE-TRAC, you can avoid introducing errors into your analysis. The potential payoffs include higher quality and faster time to market—for less money.

Rework is much more costly than building the right solution from the start. To avoid wasteful spending, RIPPLE-TRAC is an effective process for defining, capturing, prioritizing, and managing modifications to your requirements. RIPPLE-TRAC will ensure that IT focuses on the required modifications, so you can deliver what the marketplace is demanding—faster.

MAPPING A COMPONENT TO THE BUSINESS LOGIC
VIA THE OPTIONAL FLOWCHARTER. SLIDE

In summary, this logical solution USING RIPPLE-TRAC may help:

Document and reengineering - what's in the code and does it map to the business rules

Impact Analysis – What are the specifics associated with making that change

What are the complexities, resources, and skill needed to make the change happen.

Build a definitive map of what's involved.

Use as a training and exploratory tool.

Improve overall project performance and satisfaction

SLIDE FOR NEXT TWO GRAPHICS

So, if your organization assets look like the mule trains of yesteryear,

then it might be **time** to streamline your organizations reliability with RIPPLE-TRAC.