



“...the path to successful system integration...”

***The Benefits of using SyDat to support Systems
Integration***

A paper by:

Mike Duberry
Managing Director
Hawkgrove Ltd
June 2009

The Benefits of using SyDat to support Systems Integration

Introduction

There is no question that in today's business environment that systems integration is a critical activity in the successful completion of complex systems.

Lower prices and continuously more powerful computer technologies have created opportunities for organisations to develop more complex hi-tech systems. Customer demand for such systems is increasingly driven by the desire to improve performance and develop more competitive capabilities.

These increases in complexity can equate to higher project risk and integration costs, putting pressure on budgets and key resources, including project managers, developers, testers, Quality Assurance and Systems Integrators.

These pressures can lead to late delivery, budget overruns, unreliable or non-compliant system operation and potentially dissatisfied customers. No organisation would wish to be in a position where their reputation is jeopardised or where they are faced with financial damages and the potential loss of future work.

To succeed, organisations must improve how systems are developed, ensuring that they are "fit for purpose", delivered on time and to budget. A continuous improvement approach to all areas will help to mitigate these problems and help reverse the current trend of late delivery and incorrect systems operation.

Most organisations operating in today's competitive business environment are seeking to improve performance in areas such as programme timescales and wherever possible being able to mitigate or remove project risks including being able to rapidly identify and resolve any incompatibility issues.

The introduction of SyDat provides solutions to many of these problems and this paper explains how by using the SyDat technologies a different approach can be taken to:

- Enhancing the through life cycle support as systems evolve
- Facilitating the formal definition of technical interfaces between systems
- Enable the sharing of formal definitions across and between suppliers
- Guaranteeing the use of consistent definitions throughout the system development, testing and integration
- Facilitating integration testing even if supplier/system components are missing
- Providing an efficient approach to the resolution of system errors
- Reusing existing specifications and test data
- Delivering automated pass fail testing, providing significant time savings and improved accuracy

The business benefits of using such an approach for systems integrators are a reduction in project risks and delivery and a more competitive positioning with better and more accurately priced programs and improved overall financial performance hopefully leading to more satisfied customers.

Imagine if it were possible to save time and money on each project...

A systems integration technology that will typically provide the following benefits:

- An overall reduction in project risks
- A significantly reduced impact on the project from late supplier delivery
- A more efficient deployment of valuable and expensive resources
- A substantial reduction of the direct costs to produce bespoke test equipment
- An investment that can be deployed on all future projects, minimising recurring costs
- Overall reduction in direct project costs (reduced time) improving competitiveness
- Reduced risk of projects being delivered late, improving performance, reputation and bottom line
- A significant reduction of in service and through life costs

The following pages will recap on current design methods, propose a brief example project and discuss how such benefits can become a reality.

1 Traditional Project Development

1.1 Types of Project

There are many types of project, a simple analysis groups projects into:-

1. Multi Component: Systems of Systems (SoS), Distributed systems – these types of project are made up of systems (components) that reside on physically different hardware and rely on different forms of communication to achieve their collective purpose.
2. Single Component: a standalone system that serves an dedicated insular purpose that does not contribute or relate to some greater overall function (a component of a Multi Component system could be considered as such for development and testing, there still remains the complexity of integrating many of these)

The above is a simple classification that does not attempt to examine the purpose of any particular project. However the purpose of a project will also have a bearing on its complexity, which will in turn dictate the size of the integration and testing task. Consider the following as a small sample:-

1. Control: SCADA (Supervisory Control and Data Acquisition), Robotic and Hardware Management – these types of project are primarily designed to monitor, react to and control hardware that performs a critical system function. Proof of system operation can be obtained by monitoring and intelligently analysing the inputs and outputs of each system component as a collection of “black boxes”.
2. Data Exchange: Naval, Submarine, Avionics and Transport systems – these types of project all exchange information in real-time, often at high speed, possibly for display to human users, who in turn make decisions which control the overall

operation of the system, or for consumption by systems that intelligently control individual components or log data for offline analysis (“black box”) or for online diagnostics.

3. Data Exchange: Stock/Inventory Control, Financial systems – these systems exchange data at slower rates often using XML schemas

1.2 The Importance of Interfaces

In all cases, the types of project discussed above have some form of interface, the variation of which demonstrates the “black box” operation of the entire system, no matter what size it is. It can therefore be said that interfaces are a critical part of the system, for without their correct operation the system, as a whole, will not completely fulfil its purpose.

Current design methods tend to concentrate on the function of the system or systems, whilst failing to recognise that correct operation of the system interfaces is also a vital part of the system design. Additionally, the function of the system is demonstrated by what is seen on its interfaces, which provides a convenient way of proving system operation.

1.3 Current Approach to Interface Design

Traditional design methods vary greatly in their approach, from simple English descriptions encapsulated in formally controlled documents, to the use of IDL (Interface Design Language) or other formal methods.

1.4 How a project is developed, tested and integrated...

There are many design models that can be used to develop a project, we have described some of the more common ones below:-

1.4.1 The General or Top Down Model

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. There are many models, and companies tend to adopt their own variations, but all have very similar patterns. The general, basic model is shown below:-

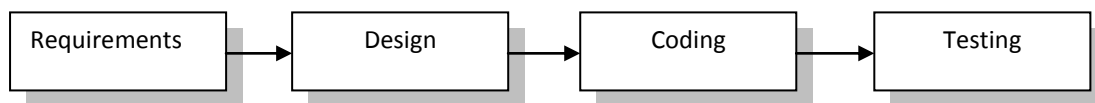


Figure 1 - General Lifecycle Model

Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced during the coding phase and is driven by the design. Testing verifies the deliverable of the coding phase against requirements.

1.4.1.1 Requirements

Business and technical requirements are gathered in this phase. This phase is the main focus for the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements. Who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system? These are general questions that get answered during a requirements gathering phase. This produces a list of functionality that the system should provide, which describes functions the system should perform, business and technical logic that processes data, what data is stored and used by the system, and how the user interface should work. The overall result is a specification for the function of the system as a whole and its performance, rather than actually going to do it which is a design action.

1.4.1.2 Design

The system design is produced from the results of the requirements phase. System architects have the ball in their court during this phase and this is the phase in which their focus and creativity lies. This is where the detail of how the system will work is produced. Architecture, including hardware and software, communication, software design (UML can be produced here) are all part of the output of a design phase.

1.4.1.3 Coding

Code is produced from the output of the design phase during the coding phase, and this is usually the longest phase of the software development life cycle. For a developer, this is the main focus of the life cycle because this is where the code is produced. Coding may overlap with both the design and testing phases. Many tools exist (CASE tools) to actually automate the production of code using information gathered and produced during the design phase.

1.4.1.4 Testing

During testing, the code is tested against its requirements to ensure the product actually meets the needs addressed and gathered during the requirements phase. Unit tests and system/acceptance tests are performed during this phase. Unit tests are performed on a specific component of the system, while system tests are performed on the system as a whole.

The above description is a basic overview of the general software development life cycle model. Now let's delve into some of the traditional and widely used variations.

1.4.2 Waterfall Model

This is the most common and classic of life cycle models and is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed in its entirety before the next phase can begin. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. Unlike the general model, phases do not overlap in a waterfall model.

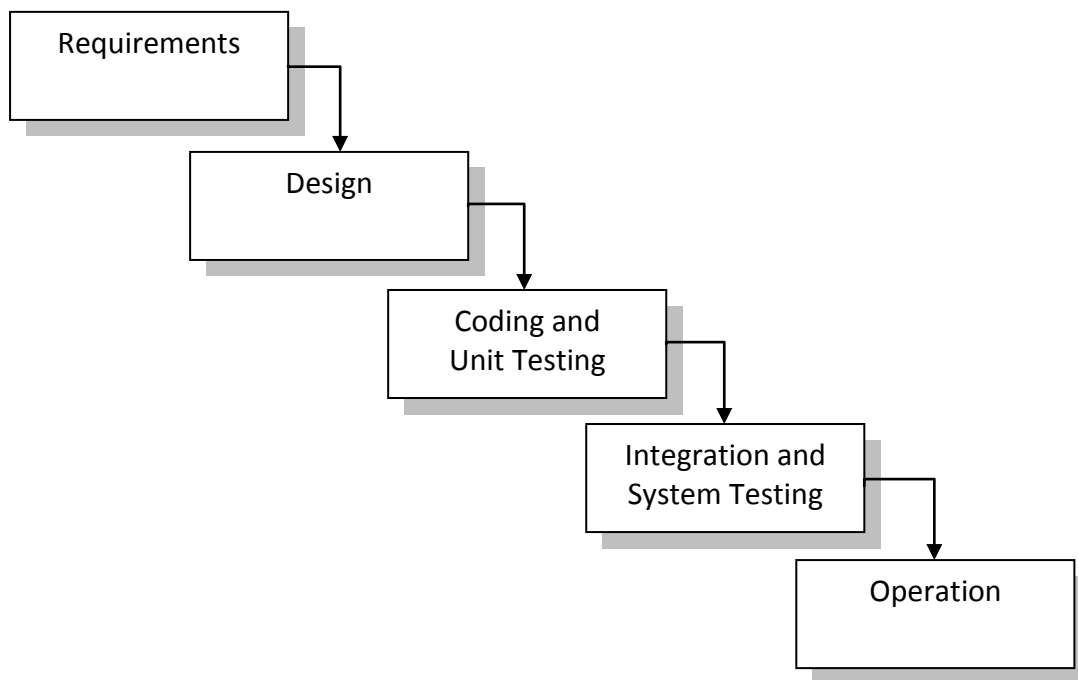


Figure 2 - Waterfall Life Cycle Model

1.4.2.1 Advantages

- Simple and easy to use.
- Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.

1.4.2.2 Disadvantages

- Adjusting scope during the life cycle can kill a project
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Poor model for complex and object-oriented projects.
- Poor model for long term projects.
- Poor model where requirements are at a moderate to high risk of changing.

1.4.3.1 Advantages

- Simple and easy to use.
- Each phase has specific deliverables.
- Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.
- Works well for small projects where requirements are easily understood.

1.4.3.2 Disadvantages

- Very rigid, like the waterfall model.
- Little flexibility and adjusting scope is difficult and expensive.
- Software is developed during the coding phase, so no early prototypes of the software are produced.
- Model doesn't provide a clear path for problems found during testing phases.

1.4.4 Incremental Model

The incremental model is an intuitive approach to the waterfall model. Multiple development cycles take place here, making the life cycle a “multi-waterfall” cycle. Cycles are divided up into smaller, more easily managed iterations. Each iteration passes through the requirements, design, coding and testing phases.

A working version of software is produced during the first iteration, so you have working software early on during the software life cycle. Subsequent iterations build on the initial software produced during the first iteration.

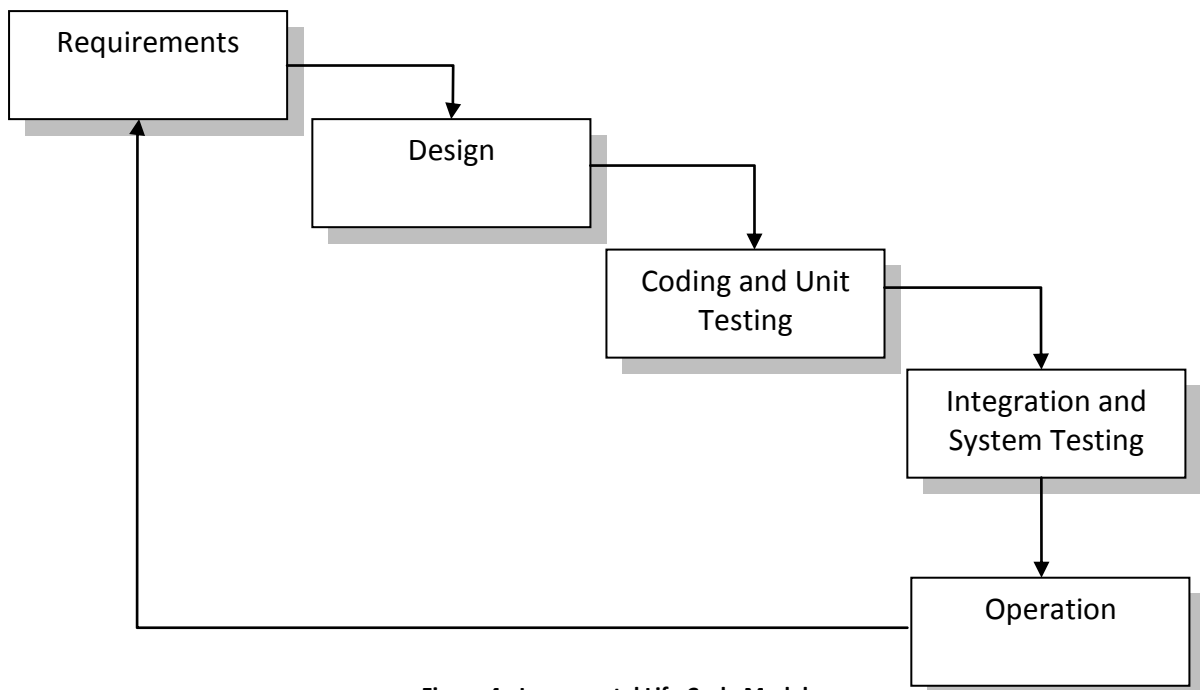


Figure 4 - Incremental Life Cycle Model

1.4.4.1 Advantages

- Generates working software quickly and early during the software life cycle.
- More flexible – less costly to change scope and requirements.
- Easier to test and debug during a smaller iteration.
- Easier to manage risk because risky pieces are identified and handled during its iteration.
- Each iteration is an easily managed milestone.

1.4.4.2 Disadvantages

- Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.

Each of the above methods includes provision for some form of unit testing, system testing and integration testing. Yet the latter two of these phases of project development requires the use of tools, simulators, representative data and the physical resources with which to perform these tasks. The later in the lifecycle a testing phase occurs, the more complex the testing in terms of tools and equipment. The definition of the test infra-structure is always late on in the project and as a result can be adversely affected by project design decisions, making it harder to prove correct system operation through testing.

1.5 System and Integration Testing

Traditionally this phase of project development includes the use of different levels of complex system simulators capable of allowing testers to verify correct operation of the system or systems, in part or as a whole.

These simulators are generally bespoke, enhanced with bespoke or COTS monitoring systems providing the facility to collect results as proof of operation. Typically the cost of these types of solution can amount to as much as 10% of the overall project cost. This cost coupled with the risk that a simulator designed from a common systems analysis may perpetuate any miss-conceptions or mistakes made during the systems analysis phase, makes the phase of system testing and integration one of the most risk laden parts of the project development.

What is the primary need to be able to integrate a system of systems? The answer is: correctly functioning interface, for, without these the systems testers and integrators cannot hope to even attempt to verify correct operation of the system as a whole. Thus, the analysis, definition and testing of system interfaces is a critical part of any project development.

Sadly, what generally happens is that one or more of the system components don't conform to their interface specification(s), maybe due to ambiguity and miss-interpretation of the specification(s). The net results are delays, extra costs, contractual problems resulting from late delivery and loss of profit. Loss of reputation and potentially, lost further work.

1.6 Typical Development Ratios

Rather than use specific values, we can look at a project in terms of the staff required to develop and test it. This allows comparison with a company by virtue of the costs of staff used on such projects. The following table attempts to describe the number of testers required when compared to the number of developers on the same project:-

Product Type	Number of Developers	Ratio of Developers to Testers	Number of Testers
Commercial Product (Safety Related)	30	3:2	20
Commercial Product (Non Safety Related)	30	3:1	10
Development & Heavy COTS Integration for a Single Client (Non Safety Related)	30	4:1	7
Government (Internal) Application Development (Non Safety Related)	30	5:1	6
Corporate (Internal) Application Development	30	4:1	7

(Non Safety Related)

The important number is the ratio of developers to testers, from this table a project manager can estimate the number of testers required to test the function of the system.

2 Example Traditional Project

Take for example a £20M safety critical transport project delivered by an external supplier.

The project development costs £18M, with a further £2M for bespoke real-time test equipment. The development of the test equipment is run in parallel with the main project and is derived from the same supplier created systems design.

At this point there is a question of independence as both the main project and test equipment are created from the same systems design. Any assumptions or misinterpretations will be carried into both streams of development. This increases the chance that the system will succeed during local supplier testing, but the lack of independence increases the risk that problems may happen following delivery of the system.

The supplier will create systems and integration test specifications that are compatible with their test equipment and cannot be re-used by the customer. In order for the customer to take delivery of the system they also create their own test environment and test specifications.

Problems found during system acceptance may or may not be as a result of poor specification of implementation, either way there will be time lost analysing who is at fault. This will be exacerbated by subsequent loss of time and money on either part whilst the problems are rectified.

Each time the customer finds a problem, the supplier must spend time trying to recreate the problem on their test equipment. If they cannot do this then they may supply releases of the project designed to isolate and diagnose the problem when it is experienced at the customer's site.

Now imagine a System of Systems with many suppliers and sub-systems, the effects of the above scenario during systems integration is a common cause for late delivery, project cost overruns and ultimately project cancelation.

The cost of building and maintaining bespoke test equipment also becomes a significant overhead as there are as many copies as there are suppliers.

3 What Happens when SyDat is introduced?

3.1 Interface Capture and Definition

SyDat is used to define the system interfaces, accurately defining the physical (hardware) and logical (data) over which communication and/or control is performed. This definition is supported by an intuitive SyDat component that provides the user with the facilities required to define system components, different types of interface, common data definitions, bespoke primitive data types, impact analysis, consistency checking and ongoing project development.

SyDat takes the interface design process further by allowing the designer to export system interface definitions for use by other SyDat components when capturing system interface data as it is transferred from one system component to another or to replace various system components allowing comprehensive system testing. Add to this, the provision of standard and bespoke facilities allowing designers to import interface definitions into SyDat any form of existing interface design, from XML, Word, Text, existing system code, and we have the means to build on the investment already made by project teams, without the need to spend countless man-hours re-entering existing interface information into SyDat. Now systems designers have a real choice if their current project needs the benefits of a tool such as SyDat.

SyDat allows the definition of the system test components on the same diagram as the system components themselves. This allows the definition of the test environment at an early stage in the definition of the project; we call this “early integration”. SyDat continues this process of “early integration” by allowing the system designer to export the completed system interface diagrams, along with the interface data definitions, into a format suitable for use by other SyDat components responsible for stimulation and monitoring. SyDat keeps the data and the physical interface separate, allowing simple migration from an RS232 interface to a LAN interface by the simple press of a few buttons.

3.2 Physical Interfaces

The fact that the physical interfaces are separate allows SyDat to use a system of agents, communicating over a LAN to allow the data to be delivered to or captured from the parts of the system that have the current focus of testing. The agents are a “loose” part of SyDat being provided using add-in technology that allows SyDat to grow both standard and bespoke physical interfaces without the need to re-release the tool. The interface to SyDat agents is open, allowing the end-user to create their own agents reflecting the need to replicate their often bespoke or secure technology. All of the real-time aspects of data delivery and monitoring are encapsulated within the agents allowing the SyDat components to deliver a feature rich, intuitive, facility aimed at providing the end-user with a comprehensive means to stimulate and analyse the system under test, all the while using the data definitions as entered by the system designer.

3.3 SyDat Components

SyDat provides stimulation and monitoring facilities. Re-using the data definitions entered by the system designer allows the generation and capture of data in both raw (bytes) and human readable (descriptive) form, simplifying the process of dealing with the data.

The SyDat components can exchange data, allowing a log file from the monitoring component to be sent to a system supplier and replayed into the product under test using the stimulation component, no need to laborious interpretation of the log file and the subsequent updated to the test environment to reproduce the log file data. The monitoring system allows complex system monitoring to take place in the form of rules, which describe how an input or inputs are transformed to produce one or more outputs. The rule is enhanced with the performance requirement of the transformation process. These facilities allow intelligent, system wide, "rules based" monitoring, which means the system tester is empowered with a tool that an isolate faults to a specific sub-system, without the need to trawl logs to find out which system is at fault.

The generation of test data is a chore. SyDat provides facilities to generate days of test data across many interfaces in minutes, the end-user can edit the test data to generate out of range values or make it part of the test generation process. Making it possible to perform a complete permutations and combinations test at the press of a button!!

One of the biggest problems during system integration is the problem of fatal faults, non performance, or late delivery of a sub-system. If that sub-system is a critical part of the overall system, then system integration is severely impacted by this. SyDat combines its powerful facilities, allowing the tester to define a cut down replacement for the missing or defective system component, enough to keep integration moving forward whilst the real component is made ready.

SyDat can import existing test specifications. As long as the format of the test specification is known, SyDat can be enhanced with a bespoke importer, ensuring that the testing investment is not lost.

4 Benefits of using SyDat

4.1 What Effect does SyDat have on the Overall Project Development?

In section 1 above, we mentioned the cost of bespoke simulators and monitoring equipment in terms of a percentage of the project budget. Also think about the need to maintain the bespoke simulator(s) and monitoring equipment, the cost and project risk of performing this task is hard to quantify particularly as the project moves from development to in-service maintenance and possibly mid-life update.

SyDat reduces the need for costly bespoke simulators and a large proportion of the SyDat system can be reused as the project progresses. Additionally, once purchased SyDat can be used again and again for every project. The system provides the end-user with the option for low cost ownership of Systems Integration test equipment that can be reused and reconfigured rather than reanalysed, redesigned, redeveloped and retested, as is the case for bespoke equipment.

Taking into account the cost of each SyDat component compared to the cost of developing bespoke equipment, it is not difficult to understand the benefits in terms of risk reduction, time saving, and significant direct cost savings of deploying the SyDat solution.

Recognising that every project and situation is different and therefore the calculated scale of benefits will also be different, however it is realistic to make assumptions in the order of 10-20% cost savings for the first project deployment and considerably higher returns on all subsequent projects.

4.2 What Happens to the Development Ratios?

SyDat reduces the risk of and takes the labour out of system/integration testing. This philosophy allows test engineers to be more productive, reducing their number and/or increasing their work output. Realistic improvements in the region of 20-30% should be achievable.

4.3 What Does SyDat do to the Integration of a Multi Component System

Imagine that all of the system component suppliers use SyDat and are supplied with a common set of system interface and data definitions.

All of your suppliers will be using the same electronic interface definitions, supported by complimentary tools allowing them to create and monitor data using common components of the SyDat suite.

Provision of such a common, definitive, feature rich, environment allows suppliers to operate safe in the knowledge that they have a representative set of interfaces, effectively each supplier is performing “early integration”, which is tending to pull the timescales to the left (shorten) as there is less work to do and a better environment in which to do it. The “early integration” also provides early confidence that the system interface definitions are correct, whilst allowing potential problems to be identified early on in the project.

The result is that when system components are delivered for systems integration, the problems experience traditionally will be considerably diminished or removed completely. Using SyDat, communications will be achieved immediately allowing the system specifier/integrator to analyse system behaviour from the outset using the same tools already being used to prove the operation of the individual components. The use of the same tools allows re-use of supplier test specifications and provides a collaborative environment allowing systems integration to proceed at a considerably improved pace when compared to more traditional approaches.

Overall this approach will reduce the time, cost, risk and in service costs traditionally associated with the development of systems that are both complex and have a long in service life.

5 Summary of benefits

Introducing the SyDat technologies into a project will typically benefit the following:

- An overall reduction in project risks
- A significantly reduced impact on the project from late supplier delivery
- A more efficient deployment of valuable and expensive resources
- A substantial reduction of the direct costs to produce bespoke test equipment
- An investment that can be deployed on all future projects, minimising recurring costs
- Overall reduction in direct project costs (reduced time) improving competitiveness
- Reduced risk of projects being delivered late, improving performance, reputation and bottom line
- A significant reduction of in service and through life costs

Currently there is no single systems integration solution offering the facilities and end to end service provided by SyDat. Systems development companies are missing out on potentially large cost and time savings that will stay with them from the day they adopt the SyDat approach. The benefits SyDat can provide will provide them “pay back” within a single project and will grow with them, allowing an evolutionary approach to ongoing systems design, integration and in service support.