

Design and Deployment of a Web Application Using SAS/IntrNet™

Thomas H. Burger, Source Consulting®, Indianapolis, IN
 Richard W. Tucker, Eli Lilly and Company, Indianapolis, IN
 John M. LaBore, Eli Lilly and Company, Indianapolis, IN

Abstract

Industry research and development increasingly relies on global operations. IT divisions now more than ever are challenged with developing an infrastructure capable of providing business solutions at this scale. SAS/IntrNet™ software offers the ability to couple powerful analysis and reporting tools with global information delivery using the Internet. Speed-to-market initiatives demand rapid implementation of this capability. We describe a methodology for the design and deployment of a web application providing end-users the capability of generating text and graphical reports.

Introduction

Corporations in the competitive climate of the pharmaceutical and biotech industry are increasingly driven to operate globally. Research and development at this scale is necessary to offset the astronomical cost of compound identification, safety evaluation, efficacy assessment, and product manufacturing. The drug development process involves mass screening, testing, analysis and reporting which may take 10 or more years to complete.

A primary objective of this process is, of course, to generate information upon which to base scientific decisions leading to a compound's discovery. The volume of information generated during compound evaluation plus interdependencies among studies results in the need to process data quickly. Important strides in this area have been made with the advent of devices for automated data acquisition and instrument interfacing. However, a potential bottleneck exists where the rate of information collection outpaces its analysis and reporting. Information systems which close the gap in information delivery for decision support reduce the risk of this bottleneck (Fig. 1).

Our objective is to describe a generalized methodology for the design and construction of web-enabled SAS® analysis and reporting systems.

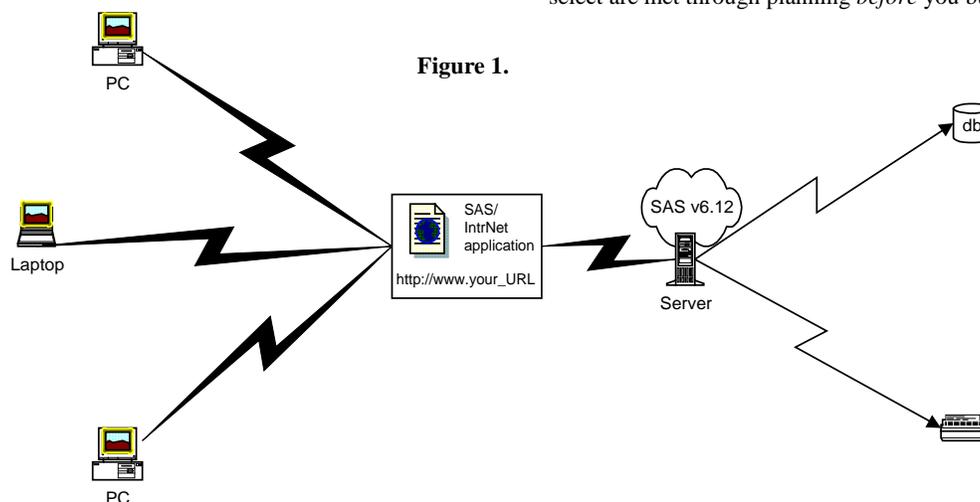
Business Drivers

A common dilemma among labs is the poor alignment of data processing with business practice. Data may exist in multiple formats or require manual intervention for standardization, error checking, subsetting, analysis or reporting. Frequently, these steps must be performed for every study. A lack of data processing automation requires manual entry or data exchange for accessibility. Further, remote access by end-users may only be a dream. At minimum, these tasks require specialized skills, are time consuming, and riddled with duplication of effort. At worst, they interrupt scheduling, work flow, and compound evaluation. Obviously, this is exacerbated as compounds move through the product pipeline and more researchers need access to data.

The time from data generation to product evaluation can be reduced by streamlining data processing to provide greater access to data, decision support, and electronic reporting. This can be achieved using the following tools:

- 1) An integrated software suite for building robust web-enabled analysis & reporting systems
- 2) A methodological framework upon which application developers can build web-enabled systems using a *rapid application development* (RAD) style approach.

The Internet provides a valuable means of global access; SAS/IntrNet™ provides an ideal mechanism for coupling powerful analysis and reporting capabilities to it. The methods we describe leverage the integration of SAS® system components while maintaining design plasticity; this leads to robust systems. Variations in the actual implementation you select are met through planning *before* you build.



In accordance with standard software development practice, we implement a Software Development Life Cycle (SDLC) approach to maximize return on development time (Sloan 1993). This front-end loads effort to the design phase, but saves time downstream by ensuring testability and extensibility (Newhouse 1997).

Analysis

Requirements are a formal definition of the criteria to be met by system functionality. **Business requirements** are system specifications written using business terminology to identify system features necessary from an operational perspective. **Software requirements** are system specifications which expand business requirements to a finer resolution while incorporating a translation into IT nomenclature. Requirements definitions interface the client's business needs with the system developer's IT solution. A clear requirements definition of functional scope is the basic premise upon which a design foundation is laid. This discussion assumes a set of business and software requirements have been defined as a basis for system development.

Design

System design provides the architectural blueprint detailing a system's processing logic and physical structure. It aids in conceptualizing what to build through schemata describing how components in the problem domain relate to one another. Its role is critical since this is where requirements are linked to system entities and where a breakdown may occur while translating requirements to code (Fig. 2). The stability, testability, efficiency and maintainability of a system rests in the architecture because it defines the context in which code is executed (Burger and Pochon 1997).

Frequently, SAS[®] analysis and reporting 'systems' are implemented as a sequence of nonintegrated or one-off programs wherein a design is lacking. While appropriate in many cases, this has limitations from a systems engineering perspective. First, efficiency of data processing is compromised. Second, robustness is incorporated into systems largely through design. Third, SAS[®] contains an array of hooks expressly intended to integrate its tool suite. Fourth, SAS[®] systems development conforms nicely to the SDLC.

Our approach to building a web-enabled analysis and reporting system rests in the need for it to be simple, maintainable and extensible. We focus on design considerations to assist us with these objectives. As a design methodology, we adapted *Structured Design Methodology* (SDM). SDM is a formal method of organization for designing a system's components and their interrelationships to best meet a set of requirements (Yourdon and Constantine 1975). It uses functional decomposition and modularity to divide complex abstractions into a hierarchy of smaller elements with clearly defined solutions. We use a CASE (Computer-Aided Software Engineering) tool to model system architecture (Burger and Pochon 1997). These methods are equally suited to macro system and AF/SCL[®] application development, as well.

After defining requirements, we identify processes of the *logical design*, as illustrated in Figure 3. By partitioning the problem domain, we clarify relationships and dependencies. Physical components and processing mechanics are addressed in separate diagrams later in design.

The *physical design* of the system is constructed by decomposing its logical components into a *Data Flow Diagram* (DFD) and *Functional Hierarchy Diagram* (FHD). The DFD models the flow of data through physical structures in the system (Fig. 4).

Figure 2.
Software Development Cycle
Waterfall Model

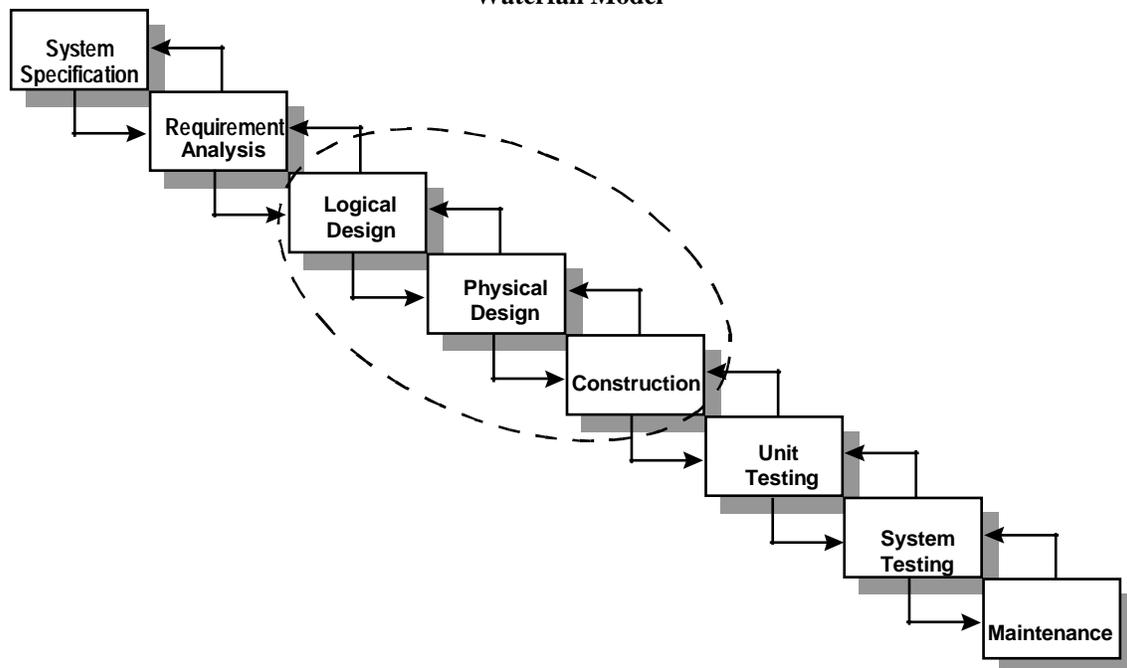
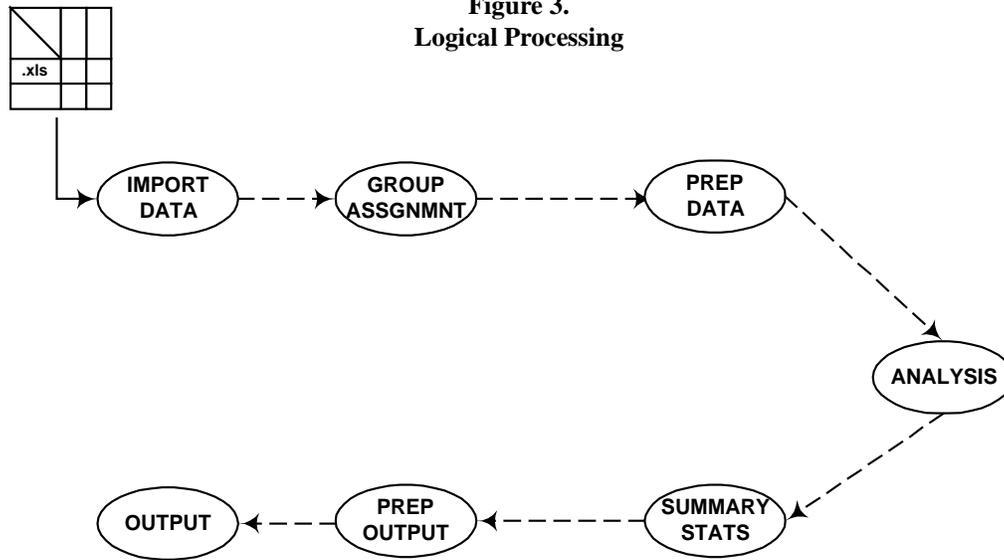


Figure 3.
Logical Processing



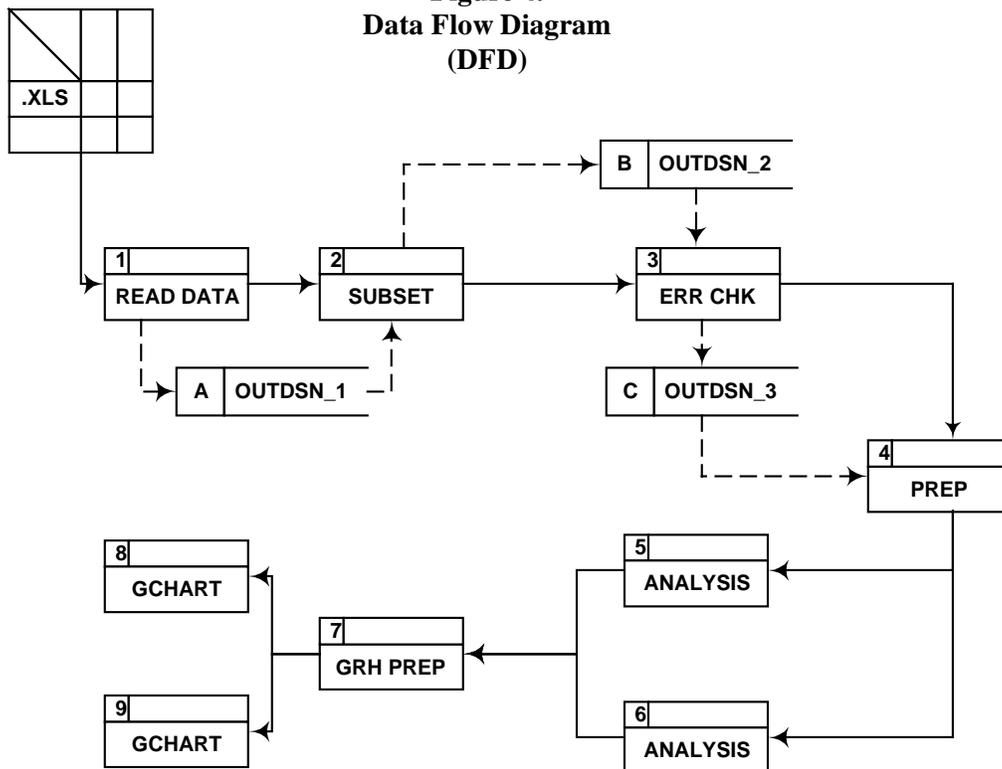
The *Functional Hierarchy Diagram* (FHD) models program organization. FHDs are horizontal decomposition charts that represent the hierarchy and calling sequence of a system's modules (Fig. 5). Decomposition limits redundancy by revealing sites of consolidation and shared functionality. Modularization permits the creation of well-structured processing thereby limiting the risk of spaghetti coding.

Modules are a logical grouping of functions whose collective effect is to perform a defined process. To impose hierarchy, externally defined macros are divided into control functions and terminal functions. **Drivers** control modules by calling

secondary drivers or sub-drivers. **Sub-drivers** control small modules by calling secondary sub-drivers or utility functions. **Utilities** are terminal functions that perform specific operations and are typically <20 lines of code (Burger and Pochon 1997).

Attention to design aids in the integration of system architecture; this contributes to stability and ease of maintenance. It also provides a mechanism to insure that requirements are met. Modularization produces programs with discrete tasks; this processing isolation results in high testability (Pochon and Burger 1998). Further, hierarchical structuring permits highly controlled processing and provides an extensible framework.

Figure 4.
Data Flow Diagram (DFD)




```

DATA _NULL_;
  IF _n_=1 THEN DO;
    .
    .
    PUT '<select name=" "&subset" '
    MULTIPLE size="3" '>';
    PUT '<br>';
    PUT '<br>';
  END;

SET IDS END=NO_MORE;
  PUT '<option value=" "&subset "'>'
  &subset;

IF NO_MORE THEN DO;
  .
  .
END;
RUN;
    
```

```

%DATAPREP(_PRPSN=WORK.PREPDATA,
          _OUTDSN=WORK.OUTDSN,
          _MACRO1=&MACROVAR)

%GRPHDATA(_DSN1=WORK.DATAONE,
          _DSN2=WORK.DATATWO)

%OUTDATA(_DSN1=WORK.DATAONE,
          _DSN2=WORK.DATATWO)

%MEND MSTRDRV;
    
```

Figure 8. displays the definition of the driver launched by the user from the second web page to load, prepare, graph and output data. As referenced in Figure 5, processing is controlled by the master driver which calls four sub-drivers, each of which calls utilities.

The overall application design is shown in Figure 9., which illustrates the following points:

- 1) The first web page, which is presented through the user interface (web browser), is constructed solely from HTML code.
- 2) The second web page is produced by HTML code created on-the-fly by a SAS® program incorporating user inputs from the first web page.
- 3) The third (and all subsequent) web pages are generated by SAS® macro modules which conditionally execute and dynamically generate HTML code.

Figure 8.

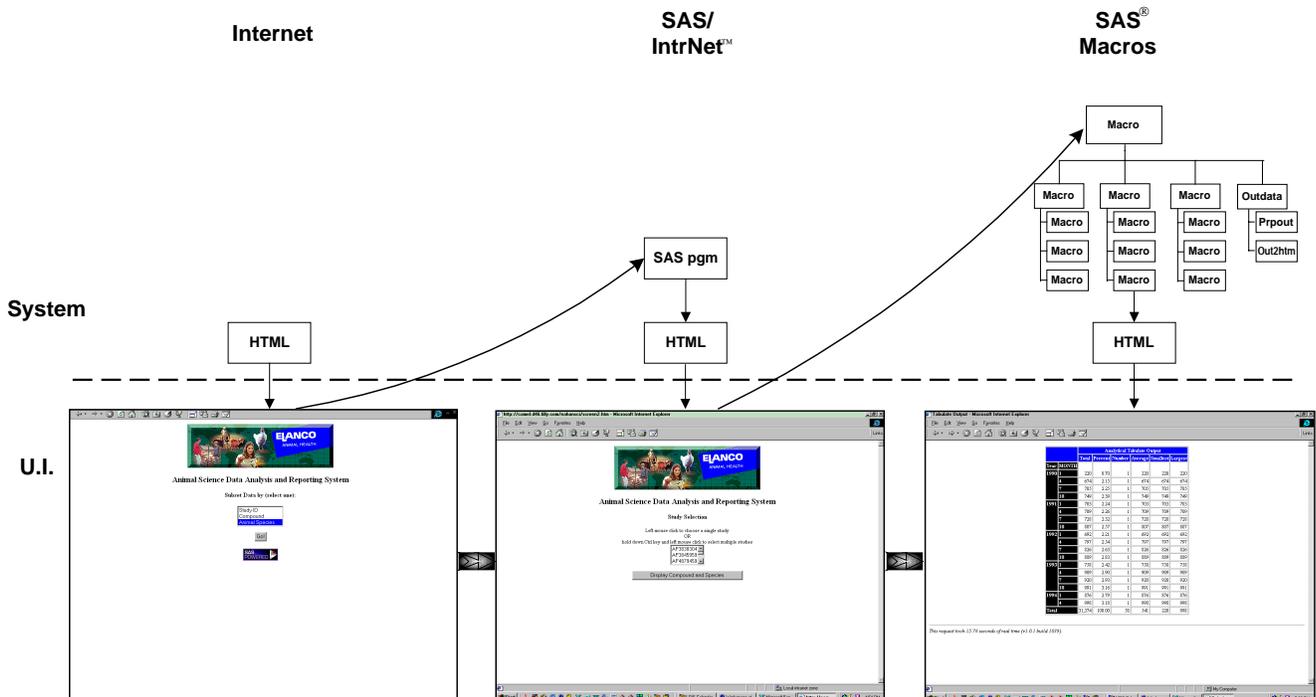
```

%MACRO MSTRDRV();

%LOADDATA(_DSN1=WORK.DATAONE,
          _DSN2=WORK.DATATWO)
    
```

SAS® Institute has a number of HTML formatting macros which can be called. In Figure 8., %OUTDATA is a call to a macro program which calls the SAS®-developed macro %OUT2HTM. Using %OUT2HTM with its default settings will produce HTML code and a subsequent web page. Further customization is quite easy. For example, if you desire red title lines on the output web page, the argument (tcolor=red) is used in the macro's invocation.

Figure 9.
SAS/IntrNet™
Application Design



Summary

We have applied a standard software design methodology to SAS[®] web-application design and deployment. Significant gains in processing simplification and interpretation can be made using a formal systems development approach. The resulting design is highly modular and easily applied to system development. CASE tools provide a useful means of representing this structure and identifying relationships. Systems created using this approach are easily maintained and highly testable. Remote access can then be coupled with on-line analysis and reporting for streamlined data processing.

References

Burger, Thomas H. and Philip M. Pochon. 1997. *Structured Design and CASE Tool Use in SAS[®] Macro Systems*. Proc. of the Eighth Annual MidWest SAS Users Group Conference. Chicago, IL.

Gill, Paul. 1997. *The Next Step: Integrating the Software Life Cycle with SAS[®] Programming*. Cary, NC: SAS Institute Inc. 384pp.

Henderson, Donald J., Edmund Burnette, Vincent DeIGobbo and John Leveille. 1997. *The SAS/IntrNet™ Application Dispatcher*. Proc. of the 22nd Annual SAS Users Group International (SUGI) Conference. San Diego, CA.

McConnell, Steve. 1993. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press. Redmond, WA.

Newhouse, Russell. 1997. *Validation and SAS[®] Programming: Benefits of Using the System Life Cycle Method*. Proc. of the 22nd Annual SAS Users Group International (SUGI) Conference. San Diego, CA.

Pochon, Philip M. and Thomas H. Burger. 1998. *Validation of SAS[®] Macro Systems*. These proceedings.

SAS[®] Macro Language: Reference, First Edition, Cary, NC: SAS Institute Inc., 1997. 304pp.

SAS[®] Macro Facility Tips and Techniques, Version 6, First edition, Cary, NC: SAS Institute Inc., 1994. 319pp.

Sloan, Faith R. 1993. *Is the System Analysis Phase Really Necessary ?* Proc. of the 18th Annual SAS Users Group International (SUGI) Conference. Miami, FL.

Yourdon, Ed and Larry Constantine. 1975. *Structured Design*. Yourdon, Inc. New York, NY.

Trademark Notice

SAS is a registered trademark of the SAS Institute Inc, Cary, NC and other countries.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Author Contact

Thomas H. Burger
Source Consulting
Bank One Tower ~ Suite 3930
111 Monument Circle
Indianapolis, IN 46140
Phone (317) 631-2900

Richard W. Tucker
Eli Lilly and Company
2001 W. Main St.
Drop Code GL50
Greenfield, IN 46140
Phone (317) 277-4896

John M. LaBore
Eli Lilly and Company
Lilly Corporate Center
Drop Code 2133
Indianapolis, IN 46285
Phone (317) 277-6387

Acknowledgment

The authors would like to thank Eli Lilly and Company[®], under whose auspices this work was performed.