Prescriptions for a Clinical Trial Automation System

Rawld Gill
ALTOVISO LLC

Abstract

Given the large quantity of information that is collected, controlled, processed, and analyzed during the course of a clinical program (one or more clinical trials), automation is required in order to achieve efficiency and quality. Yet, today (c2003) few trials attempt to utilize automation in many of the significant processes where automation technologies are available. Moreover, the industry abounds with stories of technology failures. Clearly, something is wrong.

This paper presents seven prescriptions for building a clinical trial automation system to make things right. It is further argued that alternative solutions—many of which are implemented by current technologies—will result in less than optimal systems at best and complete failure at worst.

Overview

This paper presents the following preliminaries:

1. A universal goal of clinical trial automation software is proposed.
2. A brief examination of the current market space of available solutions (c2003) is given.
3. A critique is given regarding current monikers; a new moniker is proposed for clinical trial automation systems.

Next the seven prescriptions are discussed:

1. The system must be a large-footprint system.
2. The underlying database must be a well-formed relational schema.
3. The system must provide for distributed operations.
4. Integrity facilities must be rich and easy.
5. The system must be performant.
6. The user interface must address a diverse user community.
7. The system must be cost-effective to deploy, customize, and maintain.

Goal

The goal of automation employed during drug development must be to facilitate faster, higher quality, and easier FDA submissions at the same or less cost than current methodologies. This statement has four interrelated components:

1. Speed.
2. Quality.
3. Ease of use.
4. Cost.

Copyright © 2003-2017, ALTOVISO LLC

Minimizing the time a clinical program consumes is clearly a goal for the most obvious reason: the NCE/NBA/device under development cannot begin generating revenue until it is approved. Unfortunately, since the decreased time-to-market goal has been stated so adamantly for so long by technology vendors, and unrealized (either perceptually or actually) for so long by sponsors, this goal has become cliché. Failure to achieve decreased time-to-market does not imply the goal is wrong, but rather the solutions heretofore offered are wrong. Decreasing time-to-market must be a central goal component of any clinical trial automation system since this goal will have the most effect on the financial health of the sponsor. With several major patents due to expire in the coming years[1], decreasing time-to-market is perhaps the most important strategic priority of the pharmaceutical and biotechnology industry. Finally, getting results faster than the normative case gives an additional advantage: if the NCE/NBA/device is ineffective or unsafe or otherwise unviable, it can be pulled from development earlier resulting in resource reallocation to a more promising project. Thus, systems that significantly effect the timeliness of results also effectively serve as a force multiplier.

Although self-evident, it must be said that speed without quality within the highly regulated activity of drug development is worthless. At its base, “quality” certainly requires 21CFR11 compliance. But there is much more that can and must be done. If certain quality deficiencies can be designed out of the system (and they can), then ought not such designs be preferred over competing designs that fail to offer such advantages? It is not good enough that a system was built under a proper software development life cycle (SDLC) and validated. For such statements of “quality” comment only upon construction and say nothing of design and architecture. Yet, there is tremendous preoccupation with the SDLC and its artifacts, most notably the validation package, when adjudicating the “quality” of an automation system. The reason for this is clear enough: it is easy to codify the SDLC. Design and architecture are the result of raw human creativity and imagination. There is no national standard or textbook to dictate the design and architecture of an automation system. Yet, the potential success or failure of a system is completely determined by the design and architecture of that system. For if either is flawed, no amount of enlightened development management, validation testing or glowing marketing can save the system. It will fail[i].

Thus, quality must go beyond offering evidence that the system performs according to specification. The design and architecture must, to the maximum extent possible, support the quality goals of the task at hand. In terms of drug development this implies the following:

1. The design must ensure data integrity.
2. The design must ensure data protection.
3. The design must ensure maximum system availability under varied circumstances.

[i] Examples of failed designs abound, particularly when the Internet is considered. Take, for example, the idea some years ago that the Internet would eliminate the locally installed software application. Instead, applications would be rented online. However, a thin-client, remote-server word processor is a ludicrous design. And the market has spoken: Microsoft Word and the rest are all installed as local applications.

Copyright © 2003-2017, ALTOVISO LLC

Although ease-of-use (i.e., the user interface (UI)) overlaps somewhat with issues of design and thus quality, the UI is considered separately in the goal statement for the following two reasons:

1. The UI is largely aesthetic in nature, and as such exists on a different level of abstraction than the system design and architecture.
2. The UI can be changed without changing the underlying system design and architecture.

In order to drive these points home, consider the task of building an office building:

- Architecture: The architect specifies how tall the building will be, how its volumetric space will be divided, utilized, etc.
- Design: The structural engineer specifies the steal that will be used etc.
- UI: The interior designer specifies furniture, paint colors, etc. to make the building pleasing and efficient to work in.

Architecture, design, and UI are similarly related in software:

- Architecture: The decomposition of the system into major components, the types of components specified and the communications channels between these components.
- Design: The structure of the major components; the types, data structures, and algorithms used to implement the components.
- UI: The set of components with which the end-user interacts.

The UI of automation systems used in clinical trials is particularly challenging to build as a consequence of the diverse user population. Poor UIs are not uncommon, with many systems having had dramatic success in spite of poor UIs (e.g., MS-DOS). Rigorous training and strict enforcement will result in high utilization of systems with poor UIs. But, at what cost? It is self-evident that user training is more expensive given a system with an obtuse UI when compared to a system with a less-obtuse UI (all other things being equal). However, it is likely that the loss in human productivity is the most substantive cost when considering a poor UI[2]. For when a method is difficult, illogical, obtuse, and/or otherwise unpleasant, human performance tends to decrease when compared to methods that eliminate some or all of these detractors. Consequently, a system must, to the maximum extent possible, present an easy yet powerful interface to the user.

To facilitate faster submissions without respect to cost is meaningless. Assuming a process is not resource optimized, adding resources will make it faster[ii]. But increasing the efficiency of a work process by changing the tools used and/or modifying the process itself is often a better option[3]. Indeed, efficiency of resource utilization is the raison d'être for most automation applications (think word processor versus typewriter; email versus pen-ink-paper). But no matter how dramatically a new automation system may improve the clinical trial process, the budget allocated to research and development is finite. Thus, the goal must be to maximize trial efficiency while minimizing trial cost, and to consider one without the other is naïve.

[ii] Resources are almost always scarce in for-profit businesses—pharma and biotech being no exception. But it is important to point out that attempting to increase the speed of a process beyond a certain optimum point by simply adding resources (typically, people) becomes increasingly expensive. At this point the process or the tools or both must be changed to realize further *cost-effective* improvements.

Copyright © 2003-2017, ALTOVISO LLC

Currently Available Technologies

Today there are several well-established market sectors of clinical trial automation technologies. The major sectors include:

- Clinical data management systems.
  Collect, clean, and protect the clinical data.
- Clinical trial management systems.
  Essentially, project management systems specialized to manage the workflow of a clinical trial.
- Electronic data capture systems.
  Collect case report form data at the investigator site; electronic diaries, integrated voice response systems (IVRS).
- Adverse event / safety management systems.
  Collect, manage, and protect adverse event data.
- Document management systems.
  Collect, manage, and protect electronic documents.
- Support / help desk systems.
  Provide and manage user support of automation systems.
- Project management systems.
  Manage scheduling and workflow for all or part of a drug development program.
- Statistical analysis systems.
  Provide powerful statistical analysis tools with which statisticians analyze the data collected by the clinical trial.

It is assumed that the reader is somewhat familiar with the nature of the systems listed above. While other niche and/or bleeding edge technologies exist, the list given above represents the core technologies that are both well-established and well-understood.

Note that no current technology vendor purports to provide a single system that encompasses many of these systems[iii]. Some vendors claim to integrate two (or more) technologies, but this support is always the result of integrating two (or more) established systems[iv]. The result is not a single system with high conceptual integrity, but two (or more) systems built at different times by different teams under different paradigms glued together by more or less functional interfaces (interfaces that themselves become systems). More will be said about this poor implementation strategy below.

Names, Monikers, and Buzzwords

The term “electronic data capture” (EDC) is among the most inaccurate and misleading monikers ever conceived in the clinical trial technology market space. The term is typically used to describe systems that collect case report form data at investigator sites. Note that these systems capture nothing electronically. The data is captured with the pen-ink-paper system. It is then entered into a database through the “EDC” system. In the early 1980s, such systems

[iii] Vista Clinical, Inc. (www.visitaclinical.com) has recently released a product, Panorama, which incorporates CDM, CTM, RDE, PM, DM and help desk functionality.
[iv]For example, PhaseForward’s InForm (EDC) and ClinSoft (CDM) products.

Copyright © 2003-2017, ALTOVISO LLC

were termed “remote data entry” (RDE) or “remote investigator data entry” (RIDE) systems—a far more accurate moniker. Unfortunately, the marketeers have so ingrained EDC in our minds that it is a hard name to shake.

Adding to the confusion, EDC is sometimes used to describe:

- Electronic patient diaries.
- Various telephone systems (so called integrated voice response systems (IVRS)).
- Direct data entry systems (sometimes termed bedside data entry).

All of these systems do capture data electronically, and EDC is a legitimate name for such systems.

In order to avoid confusion this paper will avoid the term “EDC”. Instead, the following monikers will be utilized:

- RDE—systems used to enter and clean data at some location other than where the central database server is located (e.g., investigator sites, telecommuting or traveling clinical research monitors, etc.).
- EPD—electronic patient diary
- IVRS—any system that collects data via interaction with a telephone
- DDE—any system that collects data directly into a database without first recording the data on paper.

Finally, the term “clinical trial automation system” (CTAS) is defined as a computer-based system that automates several global processes that occur during the conduct of a typical clinical program (i.e., one or more clinical trials).

Prescriptions for a Clinical Trial Automation System

This paper proposes seven prescriptions that a system must fulfill in order to maximize progress toward the goal given above. It argues that these prescriptions will result in dramatic progress toward the goal. It also argues that neglecting one or more prescriptions will adversely affect progress. The prescriptions do not exhaustively list all requirements for the proposed system[v], but rather present new and non-obvious thinking. An experienced system architect should be able to turn these ideas into a viable software requirement specification[vi].

Prescription 1: Large Footprint

The system must incorporate clinical data management (CDM), project management (PM) to include clinical trial management (CTM), remote data entry (RDE), document management (DM), and support management. It is argued that adverse event / safety systems are a subset of the previous systems, and are therefore de facto included. The system should not include statistical analysis.

[v] E.g., any viable CTAS must be 21CFR11 compliant and support an open architecture. Such requirements ought to be obvious to any competent architect.
[vi] Indeed, several different specifications. This paper does not intend to say that there is one “correct” way to build a CTAS, but rather that any successful CTAS must meet the requirements presented.

Copyright © 2003-2017, ALTOVISO LLC

The following is assumed as a starting point for argument:

1. Clinical data must be collected into a database, cleaned, and protected.
2. Most if not all communication activities (e.g., documents, meeting minutes, teleconferences, correspondence, etc.) that occur during the course of a clinical program must be collected, cataloged, protected, and archived.
3. The clinical program must be managed at all levels (e.g., the program level, the protocol level, the investigator level, the individual team member level, etc.).

Observe the significant overlap of each of the major components. For example, both the CDM and CTM systems collect screening data. The CDM system transfers this data to the statistical system for analysis, while the CTM system uses this data to manage enrollment. The data is, nevertheless, the same data. The common practice of faxing weekly enrollment sheets to the sponsor and then transferring this information into a CTM system (or the Excel spreadsheet-of-the-month) while recording the very same information on case report forms to later be entered into the CDM system represents redundant effort and a potential source of errors. Supposing RDE worked, screening data could be used to manage enrollment. In fact, utilizing RDE to forecast enrollment rates is actually far more reliable than any other method since, so far as analysis is concerned, the subject doesn't exist until s/he is entered into the database.

This same argument can be made when considering the management of each subject's clinical data. Resolving queries, protocol violations, lost patients, and ad hoc problems involve both management issues (read CTM) and data issues (read CDM). Why then are these two systems today considered different core technologies? Coordination with the institutional review board/ethics committee is typically considered part of CTM. Yet a major item that must be coordinated is found in adverse events—the same adverse events that are collected, catalogued, and protected by the CDM system. Since AE data is already closely controlled by the CDM system, it is a simple matter to generate an automatic periodic report to the appropriate IRB/EC. Incredibly, this functionality is often divided among *three* systems—the CTM, the CDM, and the AE system.

Viewed from another perspective, the sole purpose of a clinical trial is to fill a database with clean data from which analysis can take place and the hypothesis defined by the protocol proven (or not). Accordingly, a central purpose of CTM systems is to ensure the orderly flow of this data into the database. Using one system to measure and control the flow of data and yet another to collect, catalog, and protect the same data clearly adds a level of redundancy that causes inefficiency. Maintaining a count of the number of outstanding CRFs and queries in a CTM system requires additional analysis, data entry, and maintenance, and, in the end, will almost never be accurate. These same metrics are essentially free (via a report based upon the data) and, by definition, 100% accurate when pulled from the clinical database.

There are areas of CTM that do not involve clinical data. For example, the nature, content, and outcome of all significant telephone contact with the investigator site should be logged. If this data where treated just like clinical data (and, from a regulatory standpoint, it is just as important), then it would be collected, cataloged and protected by the same database system that serves these functions for clinical data. From the perspective of the software engineer, a record that maintains the fields {adverse-event-name, start-date-time, description, etc.} is no different than a record that maintains the fields {teleconference-site, start-date-time, description,

Copyright © 2003-2017, ALTOVISO LLC

etc.}. Once the investment is made to build a system that properly collects, catalogs, and protects clinical data, there is no reason whatsoever that that same system cannot include other classes of data.

Arguments that separating these disparate forms of data will result in more reliable and/or more secure systems are fallacious. The system is reliable and secure, or it is not. If adding additional users and data cause the system to become unreliable and/or unsecure, then the system possessed neither of these qualities to begin with. If adding more data and users degrades system performance, then the system was unscalable and hence defective to begin with. Any argument that a CDM system is incapable of augmentation to include trial management data is an admission that the system is grossly defective in one or more aspects. See Prescription 2 for further analysis of this point.

RDE systems serve primarily to enter data into the CDM, CTM, and/or safety system and provide an infrastructure to clean and monitor the data in a geographically dispersed environment. Almost all RDE systems define yet another database where the collected data is stored. Thus, almost all RDE systems require some sort of interface into the CDM, CTM and/or safety system. CDISC[4] defines an XML schema for transporting such data and certainly helps matters. But CDISC is not magic: some process/software must be defined and built to export/import the CDISC-compliant data. In the end, if the RDE database is separate from the target database, an interface must be built—yet another system has been added. Thus, under the current state of affairs, we have geographically dispersed team members (site coordinators, monitors, etc.) utilizing one system and centrally located team members (clinical data managers) utilizing another system, all operating on essentially the same data. Several layers of unnecessary complexity exist:

1. Time and effort must be expended to ensure the overlapping data in the two systems is equivalent.
2. [1] implies that an interface between the two systems must be built and maintained.
3. Some users will be forced to learn both systems.
4. Users that predominately utilize one system must communicate with users that predominately utilize the other system, thus adding a layer of contextual ambiguity to these communications. This increases complexity both from a software perspective (e.g., a query must somehow cross the system boundaries) and a process perspective.
5. The enterprise must provide user support for both systems.
6. Initial investment and maintenance costs are cumulative.

Assuming CDM and CTM systems could be successfully deployed in a distributed environment (and they can, see Prescription 3) there is no reason to maintain a separate RDE database, especially when maintaining such a database implies the significant negative effects given above.

AE / safety systems manage exactly the same data that the CDM system manages. Perhaps additional reporting and/or messaging functionality are included with such systems. But maintaining two systems that operate on the same data causes the same inefficiencies as outlined above. There is no good software engineering reason to divide safety and CDM into two systems. Assuming the CDM system is built upon a proper relational database, the functionality incorporated in safety systems is easily augmented into CDM systems. See Prescription 2 for further analysis of this point.

Copyright © 2003-2017, ALTOVISO LLC

Next, consider the overlap of CTM and PM systems. Clinical trial management is nothing more than specialized project management. There is nothing unique about clinical trials that cause standard project management techniques, processes, and procedures to work differently[5]. Many other industries require extreme rigor and are highly regulated (e.g., aerospace development, military development). Although somewhat over-simplified, in the end project management involves defining a process, measuring the performance of that process (read, collect data), analyzing the implications of the data (read, formulate metrics), and taking action to optimize the process. Since the central role of the CDM system is to collect and analyze data (albeit, clinical data), there is no good reason to introduce yet another system to serve these functions. Instead, the CDM system can be augmented to include project management data for all the reasons given above.

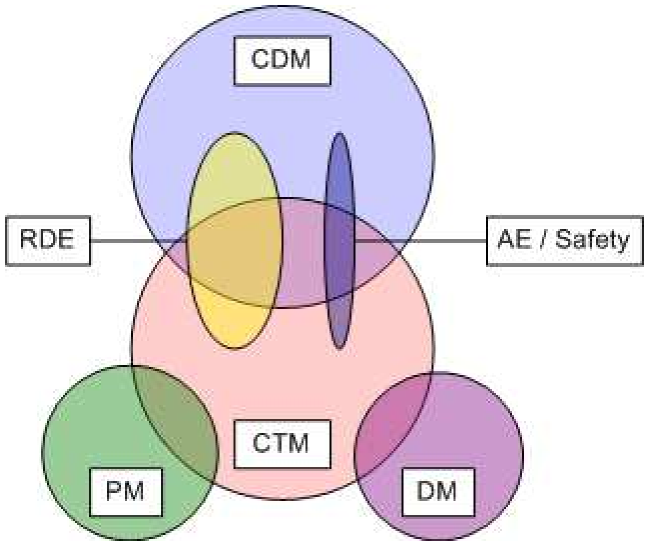
Or consider document management (DM). Typically, DM implies storing a document and any revisions, often adding a layer of control attributes to the document and the ability for a group of users to construct, review, and publish the document within a controlled and secure environment. Observe that a document is simply more data. It is unstructured data, and it is a lot of data (perhaps hundreds of megabytes). But, in the end it is just a sequence of bytes that can be stored in a database like any other data. Recent advances in database technology, hard drive capacity, and the increased address space of the typical business machine make this eminently possible[vii]. Again, it is argued that the CDM system already possesses much of the functionality required for document management and can be augmented to include document management.

All of the arguments above apply equally to support (i.e., help desk) systems. Consider an example to drive the point home. The problems reported to a help desk for a clinical trial are often diverse. They range from forgotten passwords to protocol questions[viii]. The password problem can be handled by standard help-desk software and procedures. On the other hand, the protocol question immediately becomes a controlled communication that must be tracked, logged, cataloged, and protected. But, if all support incidents are treated the same, then the many efficiencies given above are realized.

The diagram below presents a Venn diagram depicting how the various clinical automation technologies overlap. Support, which by definition intersects with every system, is not shown on the diagram.

[vii] There are two basic designs for document management systems: (1) store the document and all of its management attributes in the database, or (2) store only the management attributes in a database along with a reference to the document, which exists in some file system. Under design (2) the document management system controls the access to the document through the file system. Until recently design (1) carried some significant performance limitations; however, in most cases with modern hardware and software design (1) will now perform adequately for documents up to several hundred megabytes. Documents that require gigabytes of storage must still utilize design (2). Fortunately such documents are rare, specialized, and never distributed.
[viii] This fact is the direct experience of the author from nearly 20 years of running companies that operated help desks for clinical trials.

Copyright © 2003-2017, ALTOVISO LLC



Perhaps the most important reason to incorporate CDM, CTM, PM, AE / safety, and DM into one system is conceptual integrity. Fred Brooks states in the *Mythical Man-Month*[6],

> A clean, elegant programming product must present to each of its users a coherent mental model of the application, of strategies for doing the application, and of the user-interface tactics to be used in specifying actions and parameters.

He goes on to say (*twenty years* after the original proposition),

> Today I am more convinced than ever. Conceptual integrity *is* central to product quality[ix]. Having a system architect is the most important single step toward conceptual integrity.

How then, can a clinical trial automation system possibly hope to have conceptual integrity if it is in fact three to six systems built by different people at different times with different goals pasted together by a number of non-trivial interfaces that themselves constitute systems? The approach of integrating several “best-of-breed” systems often results in an extremely confusing and difficult-to-use system. Consider the example of Microsoft Office: Word, Excel, PowerPoint, and Outlook look and feel the same to end-users. They are comfortable to use. They all implement very similar if not identical UI paradigms. Prior to their integration other applications were frequently adjudicated as superior (e.g., WordPerfect). Note carefully our argument. It is not that Office is the best business productivity suite nor that Word is the best word processor, but rather that the degree of conceptual integrity outweighs other considerations.

Turning to clinical systems, should it not be true that a query issued by a data manager ought to be handled by the same system under the same process as a query issued by a monitor? Similarly, clinical data collection, review, approval, and locking can be accomplished under the

[ix] Italics in the original text.

Copyright © 2003-2017, ALTOVISO LLC

same paradigm as document construction, review, approval, and publication. In fact, each process that occurs within a clinical project must be defined and then the set of processes distilled into a small set of operating paradigms that can be used again and again (e.g., the “review” process; the “approval” process). Following this approach will result in a system that is easy to understand and utilize by the diverse user community that participates in a clinical program.

In summary, Prescription 1 advocates the following:

1. All data, whether clinical data, project management data, AE / safety data, documents, or support data is stored in a single data system.
2. A single system with a single operating and UI paradigm is used to interact with this data.

Such a system has the following advantages when compared to integrating disparate systems

1. There is no duplicate data; thus reconciling differences between databases is unnecessary.
2. There are exactly zero intersystem interfaces to design, build, validate, and maintain.
3. Users need to learn only one system with one operating paradigm.
4. Users communicate with each other through exactly one system.
5. The enterprise must support the absolute minimum of systems—one.
6. Investment is made in exactly one system.

Advantages [1] and [2] directly support the quality goal. Since no duplicate data exists, it is impossible to have inconsistencies between databases. Since no interfaces exist, an interface fault is impossible. Quality in these two respects is perfect, and not a single test was run. This is an example of how good design can advance quality. Advantages [3], [4], and [5] directly support the ease of use goal. All of the advantages result in lower total cost of ownership.

Prescription 1 states that statistical systems ought not be integrated. The reasons for this advice is as follows:

1. Statistical work is accomplished on a “dead” database (i.e., the data is not changing). This is quite different from all of the other kinds of data mentioned so far which is constantly changing and/or growing throughout the course of the project. (In the discussion that follows, the “dead” database is termed the analytic database and the “live” database is termed the operational database.)
2. Data transferred from the operational database into the analytic database is processed during the transfer in a way that facilities analysis. For example, highly *denormalized* tables are common[7].
3. The operations executed on the analytic database are quite different from those executed on the operational database. The former are often computationally intensive statistical functions whereas the latter are relational algebra.
4. A very small number of users (typically one or two) interact with the analytic database. Further these users are almost always co-located; thus no distributed issues exist.
5. Almost all statistical analysis utilizes SAS.

Copyright © 2003-2017, ALTOVISO LLC

For these reasons, statistical systems are nearly orthogonal to the other systems heretofore discussed. The state of the art does not support heavy statistical analysis functionality on top of operational data[8]. Although many of the advantages would apply to statistical systems *if we could operate from the same database*, we cannot; therefore, the advantages are moot.

Prescription 2: Well-Formed Relational Schema

The system must store data in a relational database management system; the database must define a Boyce/Codd normal form (BCNF) relational schema that reasonably models reality. This statement says quite a lot and has far-reaching implications:

1. The data must be stored in a relational database management system.
2. The database schema must be relational.
3. The database schema must be in BCNF.
4. The database schema must reasonably model reality.

While these statements may seem obvious and trivial, they are anything but. In fact, *most current (c2003) clinical automation systems fail conditions [2] through [4].* Owing to the popular *misconception* that a relational database is just a bunch of tables, relational database theory is frequently misused and abused, and then blamed for being inflexible or slow. It should be noted that all of the top relational database servers are perfectly capable of storing non-relational databases[x], often are so used, and that doing so grossly cripples the resulting system. Thus to say, “the system is built on an Oracle / DB2 / etc. database” **is not** equivalent to saying the system is relational. In order to understand why, and why this is important, a brief review of the relational model is in order.

The relational model consists of the following five components[9]:

1. An open-ended collection of scalar types.
2. A relation type generator.
3. Facilities for defining relation variables of such generated relation types.
4. A relational assignment operation for assigning relation value to such relation variables.
5. An open-ended collection of generic relational operators for deriving relation values from other relation values.

A relation type can be visualized as a table that has the following qualities:

1. The table has zero or more columns.
2. Each column has a heading; the heading gives the name of the column and the type contained by the column. Each row for a given column stores a value of *only* the given type.
3. The table contains zero or more rows; with no duplicate rows.
4. The table heading (i.e., the aggregate of the column headings) defines a predicate (a truth-valued function).
5. The body (i.e., the rows other than the heading) define a set of known values for which the predicate is true.

[x] For example, most of these products allow tables to be defined in a way that allows duplicate rows, which breaks the relational model.

Copyright © 2003-2017, ALTOVISO LLC

Given these definitions, the relational model allows the definition of relation types, the declaration of relation variables (relvars) of the types defined, and finally manipulation of the relvars with the relational operators to create yet more relvars of perhaps different relation types. Thus the relational model is closed (i.e. the input and output of any relational operation is a relvar). The property of closure is quite powerful, since, owing to closure, the output of one relational operation can be used for input into another. Finally, *all information content contained in the database is contained only in relvars* (The Information Principle).

There are many advantages to storing data in a relational schema[10]. Chief among these are:

1. The model has more than thirty years of rigorous scientific research behind it. Consequently, it is well understood. It *provably* behaves *predictably*.
2. The operators are based on predicate logic, which (again) is rigorously defined.

And finally, the most important advantage:

> ***The relational model allows questions to be easily asked of the database that were not anticipated at the time the database was defined.***

The sum total power of a relational database cannot be overstated[11]. It is, without question, the most important advance in information theory during the digital age. Unfortunately, the subtleties of properly implementing the model have resulted in many systems failing to realize their full potential. Further explanation of the relational model is beyond the scope of this paper. Several references are given in the endnotes.

The final two statements—that the schema is BCNF and reasonably models reality—result in a much more reliable and predictable system for both system designers and end users. Requiring BCNF results in eliminating certain update anomalies[12]. Requiring the schema to reasonably model reality greatly increases the likelihood that end-users (e.g. data managers and database administrators) will understand the schema and are therefore much less likely to misuse it.

All of the advantages of Prescription 2 result in furthering the fundamental goals of quality, ease-of-use, and cost.

In order to complete the argument, consider alternatives.

First, consider systems that define a single table in which all data is stored (several other tables define how the single massive table is interpreted). This storage methodology has come to be known in the industry as the “tall-skinny” (i.e., a few columns and many rows) or “hyper-normalized” table structure. The design has several drawbacks, the most important being ***it is not relational***. For, when examining the single data table one of three conditions must exist:

1. If the single table maintains one column to hold the actual data values, then the type of the single data column is not well defined (since it must contain many types).
2. If the single table maintains one column for each type supported, then the existence of a value in several columns is dependent upon the key, since only one column—the column of the proper type, given the row—actually contains data.
3. If neither [1] or [2] is true, then the database contains only one type.

Copyright © 2003-2017, ALTOVISO LLC

Any of these conditions break the relational model. The relational operators will not work properly and the closure property is lost. Further, augmenting the schema to include additional types (e.g., a document type) is problematic at best. For example, augmenting the data table to include a binary long object (BLOB) data type that could store a document would cause both retrieval inefficiencies in time and storage inefficiencies in space. Furthermore, the schema does not model reality, but rather models the system designer's physical view of storage. At best the system designer must build a middleware layer to provide the user (i.e., data managers) with an external view of the data that is relational; at worst this is impossible. Thus the schema is difficult to understand by end users that need to query the data in ways not anticipated when the system was built (e.g., anybody that wants to generate an ad hoc report). Finally, such schemas are not compatible with established and proven tools that can be used to manipulate relational databases. For example commercial report writers (e.g., Crystal Reports) cannot hook up directly to the data. Either middleware or a custom report generator must be written. This is a good example of poor design decisions leading to more opportunities for program fault and consequently low quality.

Next, consider so-called "XML" databases. Note that no data model has been defined or published for such databases (for those who disagree, what are the operators?). Some claim that XPath is a new data sublanguage for XML, and that XML combined with XPath provides an alternative model to a relational schema combined with (e.g.) SQL. This is nonsense. As a starting point, consider the following description from [Harold 2003][13].

> *An XML document is a tree made up of nodes. Some nodes contain one or more other nodes. There is exactly one root node, which ultimately contains all other nodes. XPath is a language for picking nodes and sets of nodes out of this tree.*
>
> And, earlier,
>
> *XPath indicates nodes by position, relative position, type, content, and several other criteria.*

These descriptions imply the following:

1. An XML document can be considered a hierarchical data structure.
2. XPath is a tool used to navigate and prune a particular XML hierarchy.

For the moment, suspend reality and assume that XML documents have some semantic properties that are common among all XML documents. Note carefully, that the fact that any well-formed XML document can be parsed *does not imply* a set of common semantic properties, but rather implies a common *structural* property. Given this assumption, one could map XML + XPath onto the hierarchic data model. But, the debate as to which model—hierarchic or relational—is superior has been settled[14]. Relational won. First, querying hierarchic databases implies navigation whereas querying relational databases implies computation. Put another way, one must specify *how* to get the answer (i.e. how to execute the query) under the hierarchic model whereas one specifies the *question* under the relational model. Clearly, *how* to calculate the answer to a question is more difficult than just formulating the question. For example, the user must understand how to navigate the database in order to specify "how" (the relational model forbids any concept of navigation). Now, lets restore reality: an XML document (loosely analogous to a table in the relational model in the best case or an entire relational

Copyright © 2003-2017, ALTOVISO LLC

database in the worst case) *has no semantic properties other than the ad hoc semantics that the authors of the document agree upon*. It does not matter that the authors may be a standards organization (e.g., CDISC). The contents of any XML document imply exactly and only the semantics the authors decided when defining the tags et cetera. Contrast this to the relational approach: every relation, no matter who or what defined it, *gives the semantics of a single, well-defined predicate*. Again, the rules of predicate logic can be used to *calculate* conclusions implied by a set of relations. Thus, "querying" a relational database is ultimately based on the provable theorems of predicate logic. Since the input to any XPath "query" has no formality, and is, therefore, ad hoc, *so must the output be ad hoc.* Further replaying the debate[xi] between hierarchic and relational models is beyond the scope of this paper. The interested reader is referred to the references.

Note: The preceding remarks are *not* intended to imply that XML, XPath, or CDISC are wrong or bad is some way. Each of these technologies is very powerful when used appropriately. Specifically, CDISC is a very important, good, and proper when used as a communications medium to move clinical information between two disparate systems. However, none of these technologies are replacements for a well-formed relational database, and to use them as such will result in disaster.

Finally, consider so-called "object" databases or "object-relational" databases. These topics are covered in depth in *Foundation For Future Database Systems*[15]*.* "Object" databases suffer problems similar to "XML" databases. Papers have been written, but they are considerably flawed. Take, for example, the following from *The Object-Oriented Database Manifesto* [16]

> We hope that, out of the set of experimental prototypes being built, a fit model will emerge. We also hope that viable implementation technology for that model will evolve simultaneously.

It should be self-evident that "hope" is an especially poor basis for a design. As for "object-relational" databases, they are nothing more than a relational database with an extended type system. As such, when/if they become available, nothing within these prescriptions prohibits or requires their use.

Prescription 3: Distributed Architecture

By their very nature, clinical trials are distributed over a large geographic area:

- Investigator sites are located at different locations than sponsors.
- Monitors typically spend more than 50% of their time traveling.
- Medical monitoring, project management and clinical data management are often *not* co-located.
- Laboratory results originate from different locations than other clinical data.

Consequently, any viable CTAS must support a distributed architecture. The goal of such a system is clear: the system must behave exactly the same at any location for any given user.

[xi] Interesting historical note: this debate was actually called the "Great Debate" at the time—1974! See reference 14.

Copyright © 2003-2017, ALTOVISO LLC

The phrase “behave exactly the same” has been distilled into 12 objectives[17]. They are defined as follows:

1. **Local autonomy**: each individual site is autonomous and does not rely upon any other site.
2. **No reliance on a central site**: as stated.
3. **Continuous operation**: the system operates continuously at every site; notably, planned shutdowns are never required.
4. **Location independence**: the actual physical storage location of the data is transparent to the user.
5. **Fragmentation independence**: the system is capable of dividing a relation variable into pieces that reside at different physical locations, and this division is transparent to the user.
6. **Replication independence**: the system is capable of storing copies of relation variables at different physical locations, and this redundancy is transparent to the user.
7. **Distributed query processing** (e.g., a SQL query, *not* a data management “query”): query execution may access data at different physical locations, and this is transparent to the user.
8. **Distributed transaction management**: transaction management (recovery control and concurrency management) functions properly when the data exists at different physical locations.
9. **Hardware independence**: the system operates on different hardware platforms.
10. **Operating system independence**: the system operates on different operating systems.
11. **Network independence**: the system operates on different networks.
12. **DBMS independence**: the system operates on different vendor RDBMS engines.

It is not claimed that all of these objectives are mutually independent, exhaustive, carry equal weight or are possible to achieve[18]. Rather, they serve to structure analysis.

Local autonomy is critical. Put another way, it is critical that certain users be able to work disconnected from any network. Monitors in hotel rooms, airports, and on airplanes present the most succinct example. While Internet connections are increasingly available, reliable, and fast, they are not universally so. Since clinical trials are often global in nature, any *strategic* system *must* be prepared to operate in parts of Europe, the Middle East, Latin/South America and other locales where the Internet is not as advanced as in the United States. Failing to recognize this will result in investing in a system that *dictates* degrading to paper-based trials when operating in these and other problematic areas. Employing a system that requires a high-quality Internet connection relegates monitors to a significant percentage of downtime while traveling.

No reliance on a central site carries all of the implications given above. Additionally, disaster recovery procedures can be implemented far more reliably if this objective is achieved. Most systems rely on a set of central servers with which client machines communicate. If this set has a cardinality of one, then losing that single server results in affecting 100% of the client population. On the other hand, architectures that do not require a single central server are far more fault tolerant. In the event a server fails, other servers can pick up the load. Further, architectures that support mirrored servers allow redundant data centers to be concurrently operated. In the event an entire data center is lost, the redundant data center can immediately and transparently pick up the load.

Copyright © 2003-2017, ALTOVISO LLC

Continuous operation is more or less important depending upon how many trials are supported by the same server cluster. If a single trial is implemented on a single server, occasional planned shutdowns for short periods of time will have little consequence. However, once the enterprise scales to deploying tens or hundreds of trials from the same server cluster, the "occasional" shutdown also scales and becomes frequent shutdowns; and this condition is not acceptable. Thus, database maintenance, mid-study changes, etc. must be executable without interrupting server availability.

Fragmentation and replication independence, distributed query processing, and distributed transaction management are low-level design details, and, simply put, must work correctly. For if any of these objectives fail, the database will become more or less corrupt. Some of these objectives can become non-issues by design choice. For example, a system could simply disallow a relation variable fragment to be distributed remotely; however, such decisions may (or may not) have severe performance implications. Clearly, the system architect must address these issues with a well-thought-out design. This design should be reviewed by a competent authority on behalf of any enterprise acquiring a distributed clinical trial automation system. More will be said about the concept of replication below; however due to the highly technical nature of these topics, interested readers are referred to the references in the endnotes[19].

At first glance, hardware and operating system independence seems unimportant owing to the fact that user systems will almost certainly be Wintel platforms. However, there are alternatives when considering the servers, and the decision to necessarily restrict deployment of any strategic system to Wintel platforms can have significant cost, security, and reliability consequences. Certainly cost, security, and reliability are all factors that have resulted in the fact that today the majority of current web servers are Unix/Linux-Apache platforms rather than Windows-IIS platforms[20]. There are certainly valid reasons to choose Sun over Wintel. A cost-benefit analysis of the various platforms available is beyond the scope of this paper other than to say: if a particular system is restricted to a particular platform, the analysis is moot; you are stuck.

All of the arguments given above apply to DBMS independence. However, the cost differential can be very significant and since it may apply to every user—not just the servers—it is worth closer examination. If the system requires a local installation of a DMBS system (and, any occasionally-connected[21] client will), then the cost of this license must be factored into the cost of each local system. Today, Microsoft offers the personal edition of SQL Server (termed the Microsoft Data Engine) free of charge. By comparison, there is no such offer from Oracle. Oracle 9i Person retails for 400 USD[xii]. This can represent as much as 25% of the platform cost.

Summarizing Prescription 3:

1. Local autonomy is critical and required.
2. Reliance upon a central site ought to be considered when formulating disaster recovery plans.
3. Continuous operation is required for heavily loaded systems; it is less important for lightly loaded systems.

[xii] As given by www.oraclestore on 13 JUN 2003.

Copyright © 2003-2017, ALTOVISO LLC

4. There are a series of low-level details that must be carefully designed. These should be competently audited before acquiring a system.
5. There are several hardware, operating system, and DBMS tradeoffs that should be considered when acquiring a strategic system. Generally, this analysis is routine.

Of these, item [1] is the most provocative for the following reason: *insisting on local autonomy necessarily disqualifies constantly-connected (to a remote sever), browser-based systems* (i.e., web-based systems). It also implies a local database will be maintained on user computers and that this database must be synchronized with some other database (typically a central server) occasionally. The industry has given the monikers "on-line/off-line" and "hybrid system" to such architectures[22]. Further, note that many so-called "web-based" systems are notoriously slow even when given reasonably fast connections. Finally, it is argued that, *assuming no configuration control* is applied to the client computer, *web-based system are slower, insecure, unreliable, and not in compliance with 21CFR11*. These arguments follow.

It is self-evident that a web-based system must be connected to the web in order to function. Such systems necessarily result in the following lack of capabilities:

1. Monitors will be unable to work on airplanes.
2. Often, monitors will be unable to work during downtime in hotels, airports, and other locations while traveling.
3. The monitor and the site coordinator will be unable to work concurrently at the site unless two Internet connections are available—highly unlikely.
4. Sites that are located in areas where Internet connections are slow and/or unreliable will not be able to utilize the system: they will be relegated to paper. Many such locations exist in Europe, the Middle East, Latin/South America and other locations where many clinical trials take place. ADSL is not available at every location in the United States; thus; if the web-based software is unable to perform adequately over a standard phone line, these locations will also be relegated to paper.
5. Direct data entry/bedside data entry will be nearly impossible. It would require a wireless Internet connection accessible in every examination room—highly unlikely.
6. Assuming no configuration control is imposed on the client, several other very serious limitations exist. These are described below.

The key to this statement is in the assumption, "assuming no configuration control is imposed on the client". The author acknowledges at the outset that given the freedom to maintain a specific configuration on a client machine, web-based software can be quite capable (albeit, still subject to the limitations discussed above). But, claims touted by vendors marketing web-based systems that (a) "any old computer running Internet Explorer will do" and/or (b) web-based systems result in low or zero deployment cost since the client computer need not be configured or maintained *are ludicrous*. *Caveat emptor!* Consider the following:

Most web-based systems demand a certain version of a certain browser, often with certain service and/or security packs installed. If no such claim is made, then is the vendor claiming that their software has been validated on all browsers installed under all possible configurations? Historically, nearly every new Microsoft product/product version automatically installs updates to the browser. Can the vendor guarantee such updates will not break the system? Is this provable? The short answer to all of these questions is, "no." This answer may be a little simple-minded, since it is likely that no real system can be *proven* correct. Instead

Copyright © 2003-2017, ALTOVISO LLC

these questions raise the problem of relative risk. Increasing deviations from the validated configuration, increase the risk of system fault. If no control is imposed on the client machine, then there is no limit to the deviation the machine can exhibit compared to the validated configuration, and thus risk of failure increases accordingly.

As for speed, simply note that HTML and XML are extremely verbose. The opening web page on Amazon.com requires approximately 70K bytes[xiii]. Contrast this to a tightly coupled system where the client and server can exchange meta-information once, and then tokenize demands for this information. Typical token streams are faster by factors of 100 to 1000. This is important to a monitor who must flip hundreds of CRF pages every day. For those who argue that XML and ADSL have made such architectures relics of the past, consider that AOL currently employs this very architecture. Unfortunately, if no configuration control is implemented on the client, then such architectures are not possible since it is impossible to guarantee the validity of the local meta-information.

If no control is imposed on the client machine, then that machine is insecure. This statement should be self-evident, but to drive the point home, the following procedure could be used to hack the client machine.

1. Footprint the machine to determine its accessibility from the web.
2. Compromise the machine and gain execute rights.
3. Install a Trojan horse virus that watches the keyboard.
4. Retrieve the user-id and password from the client via the Trojan horse.
5. Log onto the clinical automation system using the user-id and password. The attacker now has access to the investigator’s data and can cause various levels of mischief. Even more frightening, the attacker is behind the firewall and can now go on to attack enterprise-wide systems. For those who argue that the corporate firewall will block the attack from a computer other than the investigator’s, consider (a) IP address cloaking software is openly available, and (b) some local configuration control of the investigator’s computer must have taken place in order for the firewall to be aware of the investigator’s IP address—which, of course, breaks the assumption.

For more details on how this is accomplished, consult Hacking Exposed[23]. This is exactly why best IT practices dictate antivirus software, firewalls, and good security checklists. If no attention is given to the client computer configuration, then none of these protections can be reliably implemented.

Clearly, the lack of security and reliability cause 21CFR11 noncompliance.

Since synchronization has received such bad press of late, a few words are in order. Maintaining a local database implies solving two problems: synchronizing that database with one or more other databases and securing the database from improper manipulation (unintentional or hostile). Synchronization has been viewed as problematic in the past. Indeed, at least one major EDC vendor has encountered severe data integrity problems with synchronization. It is important to separate problems due to network fault (e.g., dropped connections) compared to problems due to software fault (e.g., the synchronization algorithms

[xiii] As published 13 JUN 2003.

Copyright © 2003-2017, ALTOVISO LLC

lose or corrupt data)[xiv]. Networks faults will occur under any architecture and ought to be handled transparently. In spite of these problems, nearly everybody works with a synchronization system every day: retrieving email from a remote post office is an example of synchronization. Lotus Notes is another example of a widely available and highly successful system that uses synchronization. Thus, failure of a small number of vendors to properly implement ought not condemn the technology. And, if the system is to have local autonomy, it must include synchronization.

Securing the local database is critically important. The author has many experiences with investigators trying to break into the database to "play". Generally, these attacks are not hostile or fraudulent (although, they could be). Nevertheless, the locally installed database must be secured from all attacks through appropriate architecture and security measures. This implies the following imperatives:

1. The clinical automation system must be of client-server architecture.
2. The system server and the database server must be launched in a protected address space.
3. Configuration of the system server, the database server, and other critical resources (e.g., the system clock) must be secured using appropriate operating system functionality.

Note that both the client and server(s) can (and will, in the stand-alone configuration) execute on the same machine. This is a standard configuration and operating systems support such a configuration.

Prescription 4: Powerful and Simple UI

A multi-faceted clinical trial automation system must target a very diverse group of users:

1. Computer experts who must accomplish advanced tasks (e.g., data managers).
2. Highly experienced computer users who accomplish a high volume of routine tasks (e.g., monitors).
3. Novice computer users who accomplish a low volume of routine tasks (e.g., some (not all!) site coordinators).

Naturally, there are various shades of the profiles given above. But the point should be clear: the user community is diverse. Note also, each of these users bring different expectations to the user interface. Expert users desire a very terse interface that services demands with a minimum of interaction. They also have the time and motivation to invest in learning such an interface. On the other hand, the high-school graduate that serves as the investigator receptionist—and study keypuncher—has neither the time nor motivation to become a system expert. Thus, the interface must be terse, yet easy to use; powerful, yet simple.

[xiv] The author has over twenty years of experience building systems that synchronize clinical databases that exist at different geographic locations. In all that time (more than 500 protocols, *hundreds of thousands of synchronizations*), line connections (e.g.) have been dropped (etc.) many times; however not one byte of data has been lost due to data integrity problems as a consequence of software fault. It *is* possible to do it right.

Copyright © 2003-2017, ALTOVISO LLC

The UI is naturally decomposed into major functional components. For example, report writing represents a different UI component than does data entry. Upon inspection, it is apparent that several of the UI components support industry-standard UIs. For example, there are several industry standard UIs that may be used to issue an ad hoc SQL query to a database—data managers and database administrators will need such functionality. And no UI is simpler than the UI that is already learned and is powerful enough to get the job done. Thus, currently existing and generally accepted programming languages, database definition and manipulation languages, and security and administration tools should be employed to the maximum extent possible. They have the advantage of being supported by a much larger community than specialized tools used only for clinical trials.

Nowhere is this advice more critical than when employing specialized languages that write edit checks or otherwise customize the system. A new computer language is extremely difficult to define and implement. Specialized languages should be avoided at all costs. They result in a dramatic increase in validation requirements, and a corresponding increased risk of program fault in spite of appropriate validation efforts. Yet, any viable clinical trial automation system will require customization at one or more levels. The system must support an open architecture that allows the user (in this case the programmer) to choose a language that best fits the functionality being constructed and the skill set of the programmer. Finally, the system must support connection of customizable functionality without forcing any kind of system rebuild, for such a rebuild would break the validation.

In short, if it already exists and works well, use it. Specifically:

1. Off-the-shelf report writers should be employed; custom report writers should never be employed. This market sector is well established and the system must interface with any such report writer.
2. Off-the-shelf development environments should be employed; specialized languages should never be employed. At a minimum, the system ought to be extendible with Basic, C++, and Java. Individual enterprises may attach more are less importance to these or other languages.
3. The underlying database ought to be primarily maintained with tools made available from the DBMS vendor, not specialized tools provided by the clinical trial automation system.
4. Direct interaction with the database ought to be through generally accepted UIs—SQL, query by form (QBF), query by example (QBE), etc. The database schema should be defined in such a way that users (e.g., data managers) of appropriate experience and training are able to extract useful information from the database parsimoniously without specialized system knowledge.

Note, carefully, that the advice given above applies to components that are orthogonal to clinical trials-specific functionality (e.g., a report writer versus a clinical data management system). Further, these components are of a generalized nature and released to large markets. As such, these components are integrated into the system during system construction and imply no data redundancy nor inter-system interfaces.

We are now left with the portion of the user interface that consumes the most surface area: the data entry, review, and cleaning interface. First note that all data ought to be treated the same. Thus the UI that supports entering, reviewing, cleaning, and locking a CRF ought to be exactly the same UI that supports controlling, reviewing, editing, approving, and publishing a document.

Copyright © 2003-2017, ALTOVISO LLC

Whatever the type of data—clinical data, lab data, documents, inventory, schedules, contact lists, etc.—the UI must look, feel, and behave the same. Such a system yields a high degree of conceptual integrity and all the benefits thereof[24].

Essentially three problems must be solved:

1. Navigating to data.
2. Adding/editing data.
3. Controlling data.

While not a prescription, a tree structure—much like Windows Explorer—has been shown to be a highly effective navigation UI for two reasons. First, a tree has the ability to drill down and locate a target in logarithmic time complexity. For example, one target in 1,000,000 items can be located in six navigation moves assuming each level contains 10 items[xv]. Second, Windows Explorer is so pervasive that most users are comfortable and fluent with the interface.

In addition to a general navigation UI that can locate any object in the system, customizable navigation must be provided. This navigation must present target data in a single-action (e.g., one-click/keystroke). Consider the following examples:

1. Navigate to all the data that is ready for review by the current user.
2. Navigate to all the published documents that the current user has not reviewed.
3. Navigate to all the adverse events that are marked as moderate or worse.
4. Navigate to all the support issues that are more than 3 days old.
5. Navigate to all the inventories that do not balance.
6. Navigate to all the audit findings that have not been fixed.

Such navigation paradigms allow workflows to be created that facilitate efficient and timely action. Clearly, such workflows advance the goals of the system.

Creating a UI to add and edit data is an art form unto itself[25]. Whatever UI is chosen, this prescription requires that *all data*, no matter what its type be manipulated with the same interface.

Controlling data is *the* most voluminous activity that occurs in a clinical trial automation system. Control implies:

1. Reviewing data in a timely manner.
2. Flagging, collaborating, tracking, and resolving problems that appear in the data.
3. Discovering problems that appear in the data via automation, and then resolving these problems as given in [2].
4. Measuring and tracking the timeliness and other metrics about data.
5. Certifying data elements via electronic signatures.

Note that each of these tasks involves *data about data*. In this sense, the database has two classes of information—core data and data about core data. Needless to say, any viable clinical

[xv] Note, carefully, we are now talking about the user interface, not data storage. A hierarchy can be easily stored in a proper, BCNF, relational schema (e.g., the relation {child, parent, child-attribute-1, etc.}).

Copyright © 2003-2017, ALTOVISO LLC

trial automation system must support such a concept. Further, it is critically important this second class of data is stored in a relational schema and treated just like core data. To do otherwise prohibits use of all of the tools and techniques available that operate on relational databases.

It is *strongly* advocated that similar information control functionality behave identically, irrespective of the underlying core data to which the information control applies. For example, the UI used to flag a problem on a CRF ought to be identical to the UI used to flag a problem on a document. This is where parsimony is important…and hard:

> I have only made this longer because I have not had the time to make it shorter.[26]

The marketplace is full of products that have a different widget to execute every little area of discrete functionality, no matter the overlap. Some vendors are proud of their lengthy feature lists. This is extremely poor design. The costs of validation, reliability, training, support, and user acceptance are all increased as a consequence of such systems.

The UI should contain a single information control paradigm that applies to all information control, irrespective of the underlying data or control function. All goal components are advanced by such a UI.

Prescription 5: Integrity Facilities

Finding and resolving problems with the data represents one of the key functional areas of a clinical trial automation system. Generally, problems are discovered through three methods:

1. Ad hoc human review of the data.
2. Point of entry (POE) automation that checks the data as it is entered.
3. Batch-mode automation that checks the data after it has been stored in the database.

The industry-standard term “edit check” will be used to denote automation that executes some sort of integrity check on the data.

Item [1] involves mostly UI issues and is discussed under Prescription 4.

Since the introduction of RDE software in the early 1980s there have been attempts to fix all of the problems with the data at the source. The author has participated in projects where the sponsor insisted (against advice) on *tens* of edit checks per form. The data collected from these implementations was very clean. And the site coordinators swore they would never do another study with the system. Edit checks must be implemented judiciously. Failing this, the system will be rejected.

Still, some edit checks ought to be implemented. For years industry has searched for a silver bullet to kill this monster since building edit checks implies an entire software development life cycle is executed and this is expensive. Various specialized languages and point-and-click systems have been proposed. None have been successful in the long term. The reason seems clear: there is no silver bullet. It must be accepted that uttering the statement “if male and pregnant then flag an error” is programming. One can compose the statement in C++, Basic, the Access equation editor or the favored vendor system of the month, but in the end a series of

Copyright © 2003-2017, ALTOVISO LLC

computer instructions have been issued that will modify the behavior of the system. In terms of clinical automation systems this implies the following:

1. A specification must be written to describe the desired behavior.
2. The behavior must be implemented in one form or another.
3. A test plan must be written and executed to validate the implementation.

The key to improving integrity facilities is to accept this state of affairs and leverage existing tools that aid in program construction. Thus, integrity facilities must rely on generally available languages, development environments, and tools. For example, if edit checks can be written in C++, Basic, or Java, then the wide variety of integrated development environments (IDEs), debuggers, and test automation software become immediately available. A large labor force possessing the required programming background is also available.

Conversely, to require a built-in, specialized language limits the scope of tools to those provided by the enterprise that provides the specialized system. Furthermore, the task of building a language, the accompanying development environment and supporting tools is an enormous undertaking. The relatively small clinical trial automation market space cannot support such an undertaking. Assuming such an undertaking was attempted, the result would contain a much higher risk of software fault than if an industry-standard language was utilized. Certainly, standardized languages, employed by a wide and diverse user community, are much more likely to be stable than specialized languages of limited dissemination. Finally, requiring a specialized language implies specialized programmers, which implies additional training (and, perhaps, retention) costs.

In summary, the system ought to allow POE edit checks to be built utilizing a generally available and standardized programming language and IDE.

We now turn to batch-mode edit checks. Note that the POE edit check will often be repeated in batch-mode. Two issues arise. First, it is important not to duplicate programming effort. “If male and pregnant then flag an error” should be specified, written, debugged, and tested only once. Second, there is an impedance mismatch between POE edit checks, which execute a single record, and batch mode edit checks, which execute against a set of records. The system ought to compensate for this impedance mismatch, and this compensation ought to be as transparent as possible to the user (the programmer). Naturally all of the language issues listed for POE edit checks also apply for batch mode. Batch mode processes have the additional problem of specifying the batch. Again, standard languages and techniques ought to be employed. Assuming Prescription 2 has been fulfilled, defining batches with SQL queries is a simple matter.

Prescription 6: Performance

Performance affects ease-of use. In the extreme, a system that responds too slowly will not be used. In 1985, the author set the following performance standards:

1. 0.2 seconds response time when moving between fields contained within a single form.
2. 2.0 seconds response time when demanding a form.

Copyright © 2003-2017, ALTOVISO LLC

Nearly twenty years of experience has proven these standards to be sufficient. But, are they necessary? Incredibly, the Web has caused users to be less critical of response time. Users are conditioned to accept Amazon taking five seconds to answer a demand. However, one must recognize that spending 10 minutes surfing Amazon is quite different from spending 5 hours reviewing clinical data. Several pharmaceutical companies have accepted response times in the *tens of seconds* when deploying web-based systems over the recent years. Such lack of performance is utterly unacceptable. It is unreasonable to expect monitors to function with any efficiency on such systems.

Above all, Prescription 6 requires setting a standard and sticking to it. The 0.2 / 2.0 numbers describe a performant system. Form demand response time might be allowed to degrade to 5.0 seconds without killing the system, but beyond this, anticipate a user revolt.

Also note, that a well-engineered system can be destroyed by adding intensive custom processing on top of it. Edit checks that calculate how many angels can dance on the head of a pin ought to be executed in batch mode.

Prescription 7: Cost-effective to Deploy, Customize, and Maintain

Clinical trial automation systems are very different from most enterprise systems in that they constantly change. Most enterprise systems are specified, built, and deployed and then left to run for years without modification. This is never true with clinical trial automation systems. Each protocol is a little different and mid-study changes are a given. Although a standardized eCRF and edit check library will help greatly, some amount of continuing change will be the only constant.

Thus, changes must be expected and easy to execute and deploy. All of the previous prescriptions address this issue in one way or another. The following additional requirements apply directly to Prescription 7.

1. Configuration control must be included in the system.
2. Initial deployments and mid-study changes must be propagated to end-users under an automatic, software-based process. It must not require IT technicians to manually configure machines.
3. Software version updates must be propagated automatically; system configuration must be automatically verified.
4. Virus, firewall, security, and network configuration must be automatic and verifiable.
5. Machines must be replicable without causing great hardship. For example, if a machine experiences a gross hardware fault, technical support ought to be able to pull a new machine from inventory, synchronize with the target system—thereby automatically configuring and verifying system software and project data—and send out the machine within minutes.

One must recognize that configuration and maintenance issues do not begin and end with the clinical trial automation system. The hardware, operating system, virus protection, firewall, network, hosting service, and more may be involved. It is critical that the clinical trial system take care of itself. It is desirable that configuration, maintenance, and verification of some or all of these other components be triggerable from the system. However, there must be per-enterprise problem solving in order to maximize capability in this area.

Copyright © 2003-2017, ALTOVISO LLC

Concluding Remarks

One may ask if any single prescription is more important than the others. The author believes Prescription 2: Well-Formed Relational Schema is the most important. For all operations are built on top of this prescription and if the database is wrong, the system is doomed. Top-selling CDMS solutions available today are expensive, inflexible, and incapable of augmentation to execute other tasks; they are also built using the defective “tall-skinny” table structure. If not a scientific proof, this serves as anecdotal evidence that getting the database wrong has dire consequences. Next in importance is Prescription 3: Distributed Architecture. An inflexible distributed architecture will *never* succeed globally. *Caveat emptor!* The World Wide Web offers some wonderful possibilities, and the reader should not assume the author is “anti-web”[xvi]. But the web also presents some tremendous challenges for software used in the pharmaceutical/biotechnology industry. Understand the problems and tradeoffs before jumping. Given the foundation laid by Prescriptions 2 and 3, a system can be progressively refined to include the other Prescriptions which are all judged about equal in importance.

Any system that fulfills these seven prescriptions has the potential to achieve the goal components in dramatic fashion. However, success or failure will depend as much on the commitment of those charged with deploying such a system. It is important to keep in mind that problems will occur no matter what risk mitigation techniques are employed. And—in spite of what the folks from software quality assurance may think—reams of validation documentation may have little correlation to system reliability. This “reality check” having been disclosed, the system described in this paper can be built[27], and, if deployed, will dramatically change the way clinical trials are conducted.

[xvi] Note carefully, the “Internet” and the “World Wide Web” are two different things: the Internet is a IP network; the World Wide Web is the cumulative set of interlinked documents served by the cumulative set of HTTP servers that reside on the Internet. The author *strongly* believes in using the Internet for a majority of the distributed communications required by a clinical automation system.

Copyright © 2003-2017, ALTOVISO LLC

References

[1] *Pharma 2005*, PricewaterhouseCoopers, www.pwcglobal.com: PricewaterhouseCoopers 1998

[2] P. Bickford, *Interface Design: The Art of Developing Easy-to-Use Software*, Chestnut Hill, MA: Academic Press, 1997.

> This book strikes a balance between practical advice and academic research on UI design.

[3] B. W. Hoehm, Software Engineering Economics, Englewood Cliffs, N.J.: Prentice-Hall, 1981.

> This book details how an optimum schedule can be calculated for a software project, thus implying that there is a point beyond which additional manpower yields rapidly decreasing returns. While the curves may look different for different processes, the fundamental idea that, after a certain point, more manpower does not increase process performance is universal. After all, two men will dig a 10' ditch faster than one, but 2000 men will never dig a 10' ditch faster than 1000 men.

[4] Clinical Data Interchange Standards Consortium. See www.cdisc.org.

[5] A Guide to the Project Management Body of Knowledge (ANSI/PMI 99-001-2000), Newtown Square, PN: Project Management Institute, 2000. See, also, www.pmi.org.

[6] F. P. Brooks, *The Mythical Man-Month*, Reading MA: Addison-Wesley, 1995.

> A software engineering classic, much of the wisdom elucidated in the pages of this book can be applied to managing any type of activity. *Highly* recommended.

[7] C. J. Date, *An Introduction to Database Systems*, 7th Edition, Reading, MA: Addison-Wesley, 2000.

> This book represents definitive work in the subject area. The work "introduction" in the title is inaccurate—the book is much, much more. This is the single most important reference given in these footnotes.

[8] See 7, Chapter 21, "Decision Support".

[9] C. J. Date, *The Database Relational Model*, Reading, MA: Addison-Wesley, 2001.

[10] C. J. Date, "Why Relational?", in *Relational Database Writings 1985-1989*, Reading, MA: Addison-Wesley, 1990

[11] References 7, 9, 10, and 15 are all excellent works to drive this point home. Each of these references contains an extensive list of further references that are important.

[12] See 7, Chapter 10, "Functional Dependencies", and Chapter 11, "Further Normalization I".

[13] E. R. Harold, W. S. Means, Chapter 9, *XML in a Nutshell*, 2nd Edition, Sabastopol, CA: O'Reilly & Associates, 2002

[14] E. F. Codd and C. J. Date: "Interactive Support for Nonprogrammers: The Relational and Network Approaches". Republished (originally an IBM research report) in Proceedings of the Association of

Copyright © 2003-2017, ALTOVISO LLC

Computing Machinery (ACM) Special Interest Group on the Management of Data (SIGMOD) Workshop on Data Description, Access, and Control, Volume 2, Ann Arbor, MI, 1974.

> The debate was actually between the relational and network models; however, the hierarchic model is a special case of the network model. See 7.
>
> In addition to detailed discussion of the "Great Debate", this paper gives a good example of how to compare different data models rigorously. Such skills are required in order to wade through marketing hype when deciding on new technologies.

[15] C. J. Date, H. Darwen, *Foundation for Future Database Systems*, 2nd Edition, Reading MA: Addison-Wesley, 2000.

[16] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik, *The Object-Oriented Database Manifesto*, Proceedings for the 1st International Conference on Deductive and Object-Oriented Databases, Kyoto Japan (1989). New York, NY: Elzevier Science, 1990.

[17] See 7, Chapter 20, "Distributed Databases".

[18] Consider, e.g., if two sites never depend on one another, and consequently never connect, then there is no way the data from one site will ever be available at the other. Clearly this violates location independence.

[19] See 7, Chapter 20, "Distributed Databases".

[20] "NetCraft June 20003 Web Server Survey": www.netcraft.com, 2003.

[21] D. Fineberg, "The Occasionally Connected Computing (OCC) Model, Santa Clara, CA: Intel 2003.

[22] These monikers do not exist in the computer science literature.

[23] J. Scambray, S. McClure, G. Kurtz, *Hacking Exposed*, 4th Edition, Berkeley, CA: Osborne/McGraw-Hill, 2003.

> This procedure is derived from the book—it is not taken verbatim. For those who think they are safe, *read this book.*

[24] See 6.

[25] A. Cooper, *About Face: The Essentials of User Interface Design*, Foster City, CA: IDG Books, 1995.

[26] Blaise Pascal, Lettres Provinciales, Letter 16 (1657).

[27] Vista Clinical, Inc. (www.vistaclinical.com) has built a product (*Panorama*) that implements the seven prescriptions.

Copyright © 2003-2017, ALTOVISO LLC