What is computer software?

And what determines the sustainability and maintainability of business software?

By

James Robertson

In recent weeks I have been engaged in several discussions with clients with regard to issues of sustainability and maintainability of business computer software.

Accordingly it seemed appropriate to write a short document to outline what computer software is and what determines its maintainability and sustainability.

The critical components of business computer software – what is software?

The following items comprise the critical or primary components of a computer software solution;

1. **Conceptual design and logic that models the real world**

   A fundamental requirement for easy to use and easy to maintain software is that it must accurately model the real world.

   For this reason there is a lot of merit in making use of software that is customized for a particular situation where that situation is unusual relative to common or widespread practice.

   The conceptual design is reflected in text and diagrams which explain how the software will work in practice.

2. **Data tables (databases)**

   Data tables are simply lists of information. Metaphorically they can be compared to filing cabinets and to files within filing cabinets. Well organized filing cabinets are easy to use and to retrieve documents from, jumbled, cluttered and badly maintained filing cabinets are difficult to retrieve documents from.

   The same physical cabinets and folders can be used in both cases.

   Computer databases are exactly the same. It is possible to use the same tools and technology to create data tables which are filled with jumbled data or which are filled with neatly ordered data. It is a function of how the filing scheme is designed and operated, NOT a function of the technology.

In designing a software solution it is vital that the data tables are designed in such a way that they accurately model the real world, this is called an “Entity Relationship Diagram” or ERD and defines the relationships between different logical entities such as projects, people, etc.

In order to facilitate maintenance and therefore sustainability it is vital that naming conventions should be applied that reflect the purpose of the table, e.g. “Project_Master_File” is much easier to understand than “P01” and that the fields or columns in the tables are also named in a manner that facilitates ease of understanding, for example “Project_Number” is much easier to understand than “P#”.

When building new software a set of naming conventions should be defined before anything is created and these should be rigorously adhered to in creating and maintaining the software.

When seeking to increase the robustness, sustainability and maintainability of existing software existing conventions should be identified to the extent that they exist and then documented and rigorously applied thereafter.

3. Readability

The more understandable and consistent the conventions the easier the software is to maintain. If it is easy to read and understand the documents and the source code it is generally easy to maintain it. The less understandable the conventions and less consistent the application of the standards the more difficult the software is to maintain. When we are confronted with information that flows with our understanding we digest it easily, when it does not “*make sense*” and we have to struggle to decipher what is being said, we “*chill out*” and struggle to operate effectively.

“*I do not understand computers*” can frequently be translated to mean “*what I see here does not make any sense at all, you tell me it works but I cannot see how it works so I chose to reject it OR I chose to trust you*” – in the latter case when it does not appear to work then it is very easy to abdicate and trash the system because the “*you*” clearly cannot be trusted or does not know what they are doing.

In simple terms, most published books and formal white papers and even newspaper articles are subject to review by an Editor who reviews and revises the document to make sure that it is easily understood, easy to read, makes sense, does not accidentally communicate the wrong message, etc. Thus, easy to read books are easy to read because the author AND the editor went to a lot of trouble to make them easy to read.

I forwarded the draft of this document to a number of people whose opinions I trust and got feedback that enabled me to improve the readability and other aspects of the document.

Software documentation, comments, etc are subject to the same human constraints. The comments make perfect sense to the person who wrote them as they were writing the software but they do NOT necessarily make sense at all to a third party reading them months or years later or even the same day if they are not fully informed about what the software is supposed to do.

When creating new software the table design should be comprehensively thought through and documented BEFORE work commences on building data tables.

4. Screen forms and maintenance forms

Maintenance forms are simply the screen layouts that are used for capturing data into data tables or viewing the data in the tables in order to execute procedures, etc.

A maintenance form can be very simple or very complex, in broad terms three levels of complexity are easily recognized and impact the ease of maintenance of a screen form.

A screen form that is laid out in a way that logically follows the workflow that will be applied by a human being sitting in front of the computer will be easy to use and therefore might be dubbed “user friendly” whereas a screen form that does not correlate with practical reality will be found to be “unfriendly”.

In applying the concept “user friendly”, it is important to recognize that we live in a world of devices and machines that are not in fact that friendly. A motor car is not user friendly the first time one sits behind the steering wheel, particularly if one has never seen a motor car before. But we learn to use them nonetheless.

In designing software screen forms should be laid out in detail in a spreadsheet with comprehensive documentation of what happens on the form BEFORE any work begins to create software. These prototype forms should be reviewed with the client and refined until there is a clear understanding of how things will work. The design of the forms can also impact the design of the database tables and other elements of the software design.

Use of a spreadsheet allows major changes to be made easily and allows the human beings who will have to interact with the software to get a good understanding of the logic of the screen designs before effort is devoted to creating the forms in development technology. While there are development environments that claim to offer ease of changes to screen layouts there is still merit in laying out screens in an easy to edit environment first.

Software with well designed screen forms will be easier to maintain than software with badly designed screen forms. In this case also naming conventions, neatness of working, etc are vital.

5. Configuration settings

In all but the simplest software there will be configuration settings.

This is particularly the case in "*off the shelf*" systems that require considerable configuration.

A single configuration value or "*switch*" can cause software to behave significantly differently by turning on or off certain functionality in the software.

It is vital when using software that is designed for multiple deployment situations to fully understand the implications of configuration settings.

Badly configured software can be a nightmare to use and maintain whereas orderly configured software can be extremely powerful. One badly configured logical switch setting can make the difference between an appropriate software solution and an inappropriate solution. The problem is not the software, it is the setting.

In large integrated systems where modules post data to and from other modules the configuration can become extremely complex.

In such situations the concept of "precision strategic configuration" becomes vital. This is a subject for another memorandum as this document is focusing primarily on custom software.

6. Computations, procedures, logical working steps, etc

Throughout a piece of software computations, procedures, logical workings, etc will be encountered.

This can range from very simple "Field Entry" and "Field Exit" procedures which are executed when screen fields are entered or exited to very complex computations and procedures which are executed when certain menu options are selected or buttons on screen forms are clicked.

The design and construction of computations and procedures, as with the points discussed previously, can be done in orderly and systematic ways with clearly defined rules, standards and conventions which make them easy to write, test, maintain, etc or they can be designed and built in ways that are scrappy, with no or inconsistent standards, etc.

Ideally any computation, procedure or way of working should be written first in "*Structured English*" or "*Pseudocode*" and then the computer language instructions should be written in between the structured English which should be retained in the form of comments which explain how the software works.

Such computations, procedures and logical workings make use of variables which are simply named logical entities in which values of data are stored. As with field names and other components of software design there should be clear naming conventions so that when the programming instructions are read it is easy to understand what the computation, procedure or logical working is actually doing. This makes it easy to write in the beginning and also makes it easy to maintain.

Everything that a computer actually performs is stored and executed in "*Binary*". Binary is simply "0" or "1" (naught or one). What we see in English or other human language is simply for ease of use by human beings. Before these English words like "IF", etc can be acted on by the computer they have to be converted into binary "00110101", etc.

Thus from a computation perspective it is entirely immaterial what programming language is used to create a particular piece of software. Some languages have richer functionality for particular types of computation than other languages. Thus the language is purely for the use of the human beings who program the computer. Accordingly provided a company is willing to invest in training programmers to use a thirty year old programming language there is absolutely no reason not to continue to use thirty year old software. The only reasons to change are fashion and an unethical tendency on the part of the computer industry to force customers to change software which is working perfectly well.

Insofar as source code is therefore "*instructions for the bricklayer*" in much the same way that building drawings are instructions for the artisans building the building it is reasonable for the client to receive a set of source code subject to reasonable contractual guarantees with regard to confidentiality, commercial exploitation, etc.

Well written computer software that performs a valid and valuable task can be retained indefinitely subject to some limited constraints which have more to do with unethical conduct by the industry than they have to do with any technical considerations.

Badly written software can be a nightmare to maintain and in extreme cases it may be found easier to write new routines from scratch than to try and sort out badly written code.

There are plenty of guidelines with regard to good practice with regard to the writing of code and any formally trained software developer today should adhere to a set of standards that produce good code.

As a guideline no block of code should exceed more than one A4 page when printed in ten point font and use should be made of sub-routines / objects / procedures to break large blocks of code into easily maintained blocks that perform specific pieces of work.

Code writing conventions should also include use of indents, blank lines, comments, etc in order to be easy to read.

Well written code is easily recognized and so is badly written code. In both cases a non-technical viewer should be able to form an opinion as to the quality of programming instructions quite easily.

Configuration control during maintenance, proper recording of changes – by whom, when, for what reason, etc should axiomatically be applied.

7. Content -- validation lists, taxonomies

Creating well designed data tables, maintenance forms, program code, etc is not enough. It is necessary to capture the logic of the business in validation lists, such as "*Project Type*", "*Chart of Accounts*", etc in such a way that the logic of the business is accurately reflected.

Such content can be well designed and maintained and well structured in which case the software will be experienced as easy to use or it can be badly structured in which case the SAME software will be experienced as difficult to use.

This is in itself a vast subject, please contact me for a copy of a PowerPoint presentation on the subject of data taxonomies.

Software that is difficult to use because of badly designed taxonomies can be drastically improved by redesigning the taxonomies.

In broad terms, taxonomies that are carefully designed to make sense from a strategic management perspective will make sense at all levels of the organization. Such taxonomies will support high quality executive strategic, operational and tactical inquiry and reporting and therefore high quality decision making that supports "thrive" decisions. Thrive decisions are decisions that enable the organization to be better at the essence of what it does and how it thrives.

Axiomatically, in order to obtain strategic executive level information out of a system it is necessary to put strategic executive level information into the design of the taxonomies and code schemes. This requires a strategic architect who can operate at the executive level to engage with executives and senior managers and facilitate the discussion and resulting design with strategic executive goals in mind.

Leaving taxonomies to mid-level staff and managers will result in, at best, an operational view of the data and an operational taxonomy that will not fully support strategic decision making.

Not having taxonomies at all, such as having a chart of accounts which is more or less / sort of / roughly / maybe alphabetic and which bears no relation to modeling the real world guarantees that business users will not understand the system, that posting accuracy will be low and that at best the software will be clumsy and awkward to operate. This will result in ongoing high support costs with regard to the writing and maintaining of management reports, management reports for different users that do not agree, reports that do not reconcile and, in the financial domain, long visits by auditors and high audit costs.

Such poor configuration will frequently be evidenced by staff keeping their own statistics in Excel, battling to get answers to questions, etc. All of these items appear to point to a need to replace the system, don't they? In FACT they do NOT.

All these consequences of deficient taxonomies result in software appearing clumsy and expensive to use when the problem is with the lack of logic in the master data and validation data. Take the same software and reimplement it with exceptional taxonomies to the sort of standards that I advocate and you will achieve a high value outcome at much lower cost than buying a new system. The implementation must also be properly executed.

If the software is reasonably well designed and maintained and you are able to tailor the software to your exact needs then a high value outcome is guaranteed.

I have seen badly designed implementations with badly designed or missing taxonomies that are completely defective at a level that the implementation has to be scrapped. There is nothing wrong with the software, the problem is with the implementation. I recently advised a client to scrap and reimplement a R27 million Rand investment in one of the biggest international brand ERP systems in the world because of highly defective validation data in the Chart of Accounts and elsewhere.

This is one of the biggest reasons why your current system is likely to appear clumsy and costly and why you might be considering replacing it. If you replace it and take the same shoddy validation and master data into the new system you will produce exactly the same outcome – the new system will look just like the old system because the data, which is the heart of the system, IS the same or nearly the same.

In this regard, taking the "history" across is one of the biggest mistakes you can make. Leave the history in an instance of the old software for those inquiries you do need to make, bring the history into a datawarehouse and clean it up so you can marry it with the new data but do NOT drag all the history with you at the transaction level unless there is history that is not subject to bad taxonomies. Which happens very seldom. Dragging the history across can cause delays of months, months that could be used to build high quality history.

Content, NOT software, determines the value and ease of use of any computerized system.

8. Data

In addition to the configuration data and taxonomies there is the actual transaction data that is recorded in the software.

This data, when accurately recorded, will enable the software to work effectively and when inaccurately and incompletely recorded will cause the software to be experienced as cumbersome and clumsy.

Building rules based on street names and then using inaccurate clerks to capture street names is a recipe for disaster.

We expect all staff who drive company vehicles to have the appropriate drivers license and years of incident free experience yet people are allowed to use complex software with little or no training and without first ensuring that they can type fast and accurately. By fast I mean touch type at least sixty words per minute with at least 99.99% accuracy – the standard we took for granted for data typists and secretaries thirty years ago.

Data that is full of typing mistakes will be difficult to read and use.

9. Reports

Once data has been recorded we make use of reports to extract information.

Reports can be printed or can be directed to screen. They can take the form of sophisticated "Dashboards" or simple computed indicators.

Where sophisticated analysis and reporting is required a separate "*Data warehouse*" should be implemented in order to take the processing load off the data processing systems.

Well written software that is not combined with a rich variety of useful reports that package and present information in ways that facilitate value adding decision making is of little or no value.

As with all other aspects of software, the design and construction of reports can be done to high standards of precision with standards and conventions with regard to naming conventions, variables, etc that make the reports easy to use and maintain or they can be built to ad-hoc standards in a way that makes them difficult to maintain.

Well designed and well maintained taxonomies are a vital element in creating high value reports.

10. Decisions that lead to actions that create value

Finally, the principal reason that we use business computer software (ERP, CRM, etc – all Integrated Business Information Systems) is to facilitate better decisions that add value to the business. Yes we store data for retrieval, we flag actions and workflow that is required, we automate processes and we do all sorts of other things with business software but the final return on investment is determined by better business decisions that enable the business to thrive.

Software that is badly written and badly used in an environment where there is no expectation of value adding decision making and no accountability for quality of data, etc will be experienced as value destroying.

Well written software well used in an environment in which there is clear custody of the system and accountability for decisions made using the data in the system will be experienced as value adding.

Well written software badly used will destroy value and badly written software well used will create value.

In all aspects of computer software the value is determined by the human beings who use the system.

11. Build versus buy and retain versus replace

Easy to maintain software with a long investment life is determined by the right technology choices at the start of the life of the investment and by the right standards and adherence to standards throughout the life of the software. Well designed, well written and well maintained software can last for decades.

The bottom line is that the debate should not be about "build" or "buy" it should be about the appropriate use of appropriate elements of the technology for the required business outcome. Where there is an "*off the shelf*" solution that is a good fit to the business, then "buy" is the way to go. Where the business has elements that are very specific and very different then "build" should be the automatic choice.

Thus, if you have a system that is working but you are concerned about sustainability and maintainability, do whatever is necessary to raise standards of documentation, etc to appropriate levels. If there is a very small team of people supporting the software invest in training up an additional person or persons to be able to support the software.

Where appropriate invest in keyman insurance.

Treat the system the same way you would a factory and ensure that your investment is secure for many years.

12. It is ALL in the name? – not so

A fully functional integrated business information system comprises dozes, sometimes hundreds or even thousands of data tables, maintenance screens, executable programs, etc. One should be careful about giving such things names. They are complex factories, warehouses, etc, etc and if some element is not working correctly it should be adjusted as necessary.

Frequently the reasons for non-performance relate to the actions or inactions of human beings and not the technology itself. If we call the who system "Fred" then it becomes seductively and misleadingly easy to say that "Fred" is not working correctly when we should actually say "*there is a minor error in the way the xyz module computes abc under certain specific circumstances*".

When we give a system a name then it is only a small step to start conversations about "*is Fred the right system*" or "*is Fred out of date*" etc when from consideration of the preceding sections it will be apparent that there really is no such thing as Fred and we should focus on whatever remedial work is required to the element of the technology portfolio that is not performing as expected, including looking at what the human beings using the system are doing.

I have assisted clients with conversations like this about "Fred" where "Fred" ranges from "SAP" to some home grown system cobbled together over the years. If it is badly implemented and badly configured it will be a mess no matter what world class technology is used.

Please contact me for further information.

Dr James Robertson PrEng

30 April 2010

James@JamesARobertson.com

++27-(0)83-251-6644