CRIMSON PUBLISHERS
Wings to the Research

Archaeology & Anthropology: Open Access

ISSN: 2577-1949

Review Article

Anthropological Analysis of Technology Assets Change Paradigm Support

Mordechai Ben Menachem*

Ben Gurion University, Beer-Sheva, Israel

***Corresponding author:** Mordechai Ben Menachem, Ben-Gurion University, Beer-Sheva, Israel

Submission: Jan 30, 2018; **Published:** June 15, 2018

Abstract

Often the most fertile insights into contemporary problems come not from those in the mainstream but from the more adventurous spirits who have charted their own intellectual course. Definition of anthropology: the science of human beings; especially: the study of human beings and their ancestors through time and space and in relation to physical character, environmental and social relations, and culture. What are the anthropological aspects of information and of software-the entity that controls information (at least, certain dimensions of control)? We know that software and information are both notoriously difficult to manage well. What does that mean, why is it important, and what can be done about it? Much to the surprise of many readers, these questions are life critical issues, for almost all of humanity.

Keywords: Information

Information Management Objectives, Including Software

Information is data processed in such a manner such that it becomes meaningful for the user (consumer) of that information. Namely, data organised for semantic association. This is particularly true of "information about information" ("meta-information"). Software is a form of information; what we are talking about is information used to describe software. The IS' role is to capture data, organise by semantic association and allow meaningful uses via expressive queries. In addition, IS' are most successful when the data has human-imposed structure. Relational systems provide powerful tools because they supply "human" constraints, helping users fit it to the application domain. There exists the counter example of the world-wide-web (totally devoid of structure). However, this is not a refutation because hyperlinks are not a coherent information domain (which is exactly its strength, it is not meant to be one information system).

The spectacular advances in information-processing capabilities have generated many paradoxes. Three good examples are the following: (Table 1).

Table 1: Impossibilities.

Impossible Manufacturing	Impossible Consumption	Impossible Valuation
Information goods are produced and distributed to thousands of points instantly, at "zero cost" impossible with traditional goods.	One can sell or give the product away innumerable times and still be in absolute possession and ownership.	We spend enormous amounts of money on production and maintenance, but do not know how to value it.

Increasingly, business executives have a need to better align Information Technologies with business strategies, with organisational change and with mergers and acquisition activities. Information Technology has become inextricably woven into a basic business fabric. Despite this, there is still an endemic lack of organisational flexibility to take advantage of IT progress and the same lack of flexibility in IT to continue to simulate rapidly changing organisations. The lack of IT flexibility causes corporate pain seemingly as large as the issues it resolves.

Business Management Objectives

One important and fundamental aspect of mergers and acquisitions is a good match of the Corporations' Information Systems. IT is very knowledgeable; storing knowledge of every aspect of the business and relationships and enabling functional management of every part of the organisation, except itself (excepting IT). We still do not have high quality information systems to manage the software that manages all the other information in the business-a logical absurdity, but a fact of corporate life and a critical twenty-first century human conundrum. We use Information Systems to accurately manage assets of all functional activities (i.e. manufacturing, account management, human resources, marketing) but do not use IT to manage IT assets. For instance, until quite recently, the only way computer systems could be inventoried was by manual processes; so difficult and cumbersome as to border on absurdity.

Enterprise Software consists of tens of thousands of files, many with multiple objects and many with multiple uses. In one major insurance company, it took three people almost two years to have an inventory! We propose an automatic method to do so in weeks. Not only is the manual process too onerous to be practical, but the difficulty of auditing its' accuracy and keeping the inventory up-to-date, makes it virtually impossible to use with any real effect; there can be no long-term use, as it is inevitably obsolete by the time it is "completed." Certainly one would not expect to manage any other type of asset without an inventory. By the way, average personnel turnover is always too short for maximum effectiveness.

In another major insurance company, they were asked to assess the number of files they store in their systems. The author was told: "We have between 35,000 and 45,000 files, but I do not know what languages they are written in or who uses them." Thirty-five to forty-five thousand? That is the best guess they can come up with? What other industry would tolerate that degree of incertitude?

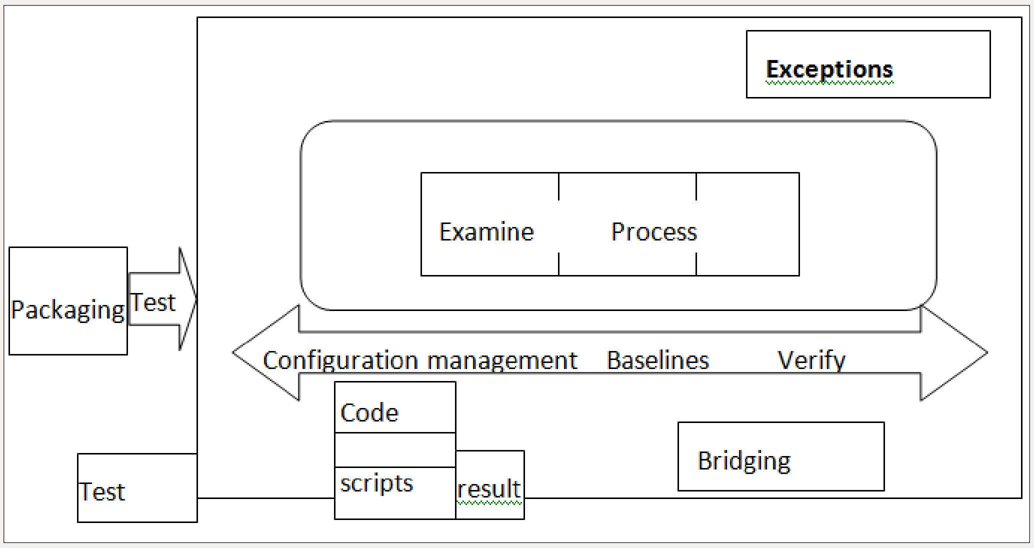


Figure 1: The Software Factory model.

Even having an inventory, this is only the first process towards the end to achieve and we have already said that it is not trivial. Overall, our objective is Management of all Information Processing Assets in the Enterprise. We call this: "Enterprise Resource Management" (ERM). Clearly, this name is a "take-off" on Enterprise Resource Planning (ERP) which has become so prevalent in manufacturing and service industries. However, just as ERP evolved over time from islands of information through stages of MRP to where it is today, so all ERM needs to evolve. The vast majority of corporations are still at the stage equivalent to islands of information. Our objective is to evolve IT in a similar manner, but much more quickly. We believe that the technologies exist to allow this to happen. We believe that the marketplace has a sufficient understanding of the issues involved, thanks to Year 2000, to understand the needs (Figure 1).

Management Structures of Organisations

The type of organisation should (and usually does) determine the way it is managed. The basic management model determines information needs. Competition, external controls (e.g., government overseers) and vulnerabilities influence this greatly. In examining successful organisations and attempting to predict what contributes to future success, it is useful and interesting to examine what different types of organisations need. It would be counter-productive to assume all organisations have identical needs or characteristics.

Clearly, organisations' goals heavily influence needs. It becomes increasing clear that a good product, an innovative strategy and charisma are no longer sufficient. With increased product technology sophistication, differences become muddled and prices no longer control purchase decisions. Increasingly, large companies have less of an inherent advantage over small ones (or visa-versa). Less of an advantage-not no advantage. Size still counts, just differently. Where advantage lies is not a large or small company, but a great one or "another" one. Customer's decisions are influenced:

A. Not by price but long-term costs

B. What really matters is not this sale but how you ensure a long-term relationship between your staff and the client's staff (i.e.,) the next sale, and the ones after that

C. Not by "we try harder" but "we solve YOUR problem"

D. What really matters is not what you have done, but what have you done for me lately. (Not nice, only true.) Customer loyalty just does not last as it used to

E. Not by what you make but what you stand for

F. Product and business cycles are getting too short for this to matter over the long-term. What really matters is your identity and purpose

G. The following shows organisation of these ideas. In each case, the "organisation" has orientation and ideas that match its needs. Their standards activities are neither "good" nor "bad." They are simply what their needs reflect (Table 2).

Table 2: Organisations' information needs.

Organisation Concept Type	Mind-Set	Orientation	Typified By
Army	control people direct events	security	60,000 MIL standards
Government	manage people control events direct information	survival	laws
Traditional "low-tech " industry	lead people manage information control ideas	profit	ISO 9000
high-tech industry	lead ideas manage information foresee events direct innovation	rapid growth	process standards seldom work

The secret to survival is adaptation to change-notice, not controlling change, but adapting. Change is inevitable controlling it is nonsense. A great company learns how to constantly adapt itself (i.e., re-invent itself!). Not only when a problem is recognised and MUST be responded to, but constantly. Organisational adaptation to change is an integral part of corporate culture. How can this be accomplished (much less, managed) without a very powerful information model? Remember that the information system reflects the organisation. Information systems need to be "infinitely" adaptable to constant change. This is not possible unless the Information Technology is managed. The degree and sophistication of this management must be as high or higher than the organisation as a whole.

Managing Enterprise Change

Change is the reality of business. Change is not something that happens, it is something that is anticipated and brought about in order to excel. Change may occur as a result of a merger, an acquisition, deregulation, new market constraints, increased or new customer focus, smart management, new technology, operational changes renewing the IT function, outsourcing a specific task etc. Whatever the reason, the process of change must be carefully managed. Whether the organisation is a reflection of its information system or the IS reflects the organisation, all indicators show that this process will intensify. All organisations will be in a state of constant renewal. The more successful companies will be those that can renew themselves in unique, forceful and creative ways. This includes structure, products and processes. This may be reflected in corporate culture, technologies used or human resources. Clearly, one of the more important aspects is how these are integrated.

One example of this is Remote Access or Tele-Work. This enables workers to access the corporate information Knowledge Base from anywhere on the globe to find and interact with it and with clients. As talent becomes more difficult to acquire, this is a prime business enabler. In 1995, Eric Vogt [1] published a seminally visionary article which envisioned employing "nomadic knowledge workers." When, for example, graphic talent is needed, but the best person for the job prefers to live and work far from the corporate scene, say in a rain forest or desert should an outdated corporate structure, based upon remains of an agrarian society, limit our ability to take advantage of these needed talents? The answer should certainly be negative. It means a new way of thinking about organisation; one must control corporate information assets (Table 3).

Table 3: A System characteristics vs goals comparison.

System Characteristic	Applications	Goal
1. National security		system integrity
2. Interactive		responsiveness
		usability
		soft fail
3. Long system life		portability
		testability
4. Human lives depend upon it		reliability
		testability
5. Corporate stability depends upon it		(write your own list here)

A Quality Environment

The reality is that software maintenance has been viewed trivially, as a "necessary evil" and not as standard operating procedure. Software maintenance always exists. Software systems are the oldest technological artefacts we deal with. It is not uncommon for organisations to continue to maintain systems developed several decades ago. It needs to be understood that the people that wrote these systems did not think that they would last more than a few years, if that. This is a major aspect of the difficulties surrounding software maintenance. The way to constant improvement is straightforward. Constant improvement is a question of the right methodology, applied with the right discipline, at the right time and by the right people, consistently.

Development is never simple and maintenance can be even more difficult. Procedures are divided into two classifications: "Operational Procedures" (for such issues as plans and control) and "Technical Procedures" (for such issues as defining requirements, specifying design and testing). Many procedures can be software supported to help the process be performed more smoothly.

The farthest-reaching difference perhaps concerns systems' requirements. Requirements acquisition and management should be an enterprise-level function. In most cases, it is not, because we do not really know how to do it. Requirements management is not the Requirements phase of the development life cycle. Requirements management is a continuous process and this is the area most 'ripe' for anthropological aid.

Organisations have large quantities of deployed software. HOW are we ever to acquire the requirements from them, from the software already deployed? We know of a case where the programmers did not like a certain management decision so they decided to "show management how things really work." They all left, taking much of the software and all of the knowledge of how it worked, with them. The salaries of the people who worked for this organisation were quite complex. No one, other than the programmers (and the software) knew how to compute pay. There was total dependency of the organisation on this small group of workers (tens, out of tens of thousands in the entire organisation). Clearly, had they such a process, fully documented of course, for ascertaining the requirements, the subsequent blackmail could not have occurred.

Development

All existing source code must be stored for Enterprise access, using a Software Configuration Management system or some derivative model. Access to this should be limited in a way that is acceptable to corporate needs, project needs and development staff [2]. All source modules accessed for update must be accessed only through the SCM/EAM system. Such access, both check-in and checkout must be logged by the system and controlled by project management. All new code modules must be created in a manner that project management retains control over naming conventions. Use of new tools must be checked by the authorised role in the organisation. Use of un-approved tools should not trivially permitted.

EAM-Enterprise Asset Management

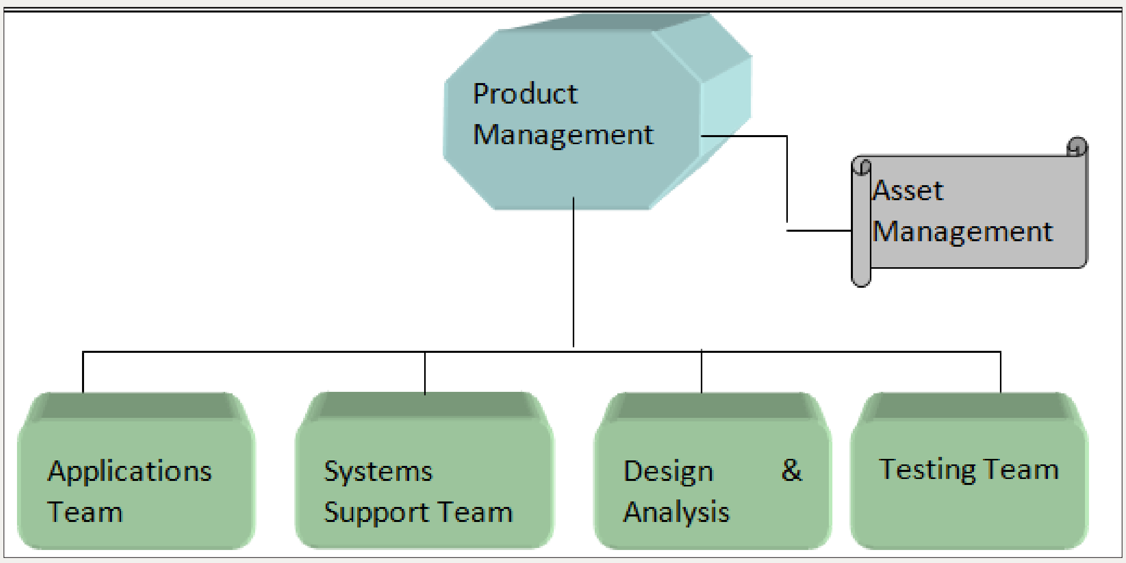


Figure 2: A possible product management model.

Basic Software configuration Management is deficient and the clearest proof of this was the effort demanded to implement it. Not technically, almost any place can buy a Commercial Off-The-Shelf system from myriad suppliers. Some have acceptable performance. These systems store existing files in a manner which is difficult to access and save the differences between the versions of each file. However, programmers do not like inhibitions; it bothers their creativity, which is what they are being paid for. We need a more advanced concept. We need to directly link files of code with other programming objects. We are starting to see some of this. All of these need to be taken into account for the project to compile into a working system. Many if not most, programming systems are either already doing this or are moving towards this state. Additionally, we need to add to this all of the word processing files, presentations and other tools' files which form parts of the development project. SCM tools were not designed for this level of multi-dimensional linkages. They were designed to store text files (Figure 2).

There are two problems. First is that systems' development technologies have really progressed much faster than the tools; and hence are really not being used as their designers intended. The second is that even if we solve the first, they are frequently viewed by developers as impairing the development process rather than enhancing it. What is the solution to this conundrum? We need a model. The Enterprise Asset Model needs to provide "natural" links between the development objects and their management needs.

Testing

The time has come to really implement system testing techniques. Lawsuits from unplanned downstream impact will kill you. Besides, pay-back is so immediate and obvious that there should not really be a need to talk about it. The author was sitting on an aeroplane flying between two cities in the United States, when I noticed that the gentleman sitting next to me was looking at the kinds of documents that only this profession creates. We began talking. It turned out he was the "ultimate" IT manager for one of the very large banks. He told me the following story.

They completed a conversion of their mortgage system. It had completed absolutely on budget and on schedule. They were very happy and very proud. While monitoring the system, as if by appointment, a client walked in to a branch and asked to pay four years in advance on the mortgage account. The system accepted the payment with no problems. Handshakes, Applause, Euphoria! About two weeks later, the same client telephoned and asked to have a printout of the account balance. The command was entered and the whole system crashed.

They had converted the code correctly but there was a mess in the tables. Almost the whole project had to be redone because what had been done was now realised to be incorrect!

Testing is no longer a choice. It is a survival imperative. It is also expensive and technically very difficult. The choice is corporate death. Not a very good choice, but we have been through that before.

Standardisation

It does not really matter which set of standards you use. You may decide upon Software Engineering Institute's CMM or on International Standards Organisation's "ISO 9000" or on Bootstrap or Trillium [3]

or SPICE or on anything else which you deem suitable. You must now begin to think in terms of being able to repeat your successes and avoid future iterations of failures. Whatever you do decide to use, we suggest that you look at the following.

A. Obtain commitment by senior management.

B. Establish an Impact and Improvement Council with director level participation.

C. Obtain management participation.

D. Secure team participation.

E. Obtain individual involvement.

F. Establish system/product/process improvement teams, with clear representatives of everyone involved.

G. Develop and direct information and software supplier involvement activities.

H. Establish the systems quality assurance activity with very clear long-range quality strategy and short-range quality plans.

I. Establish recognition and reward system.

J. Establish absolutely clear lines of authority.

Exposing Maintenance Complexities

An anthropological study of information systems' maintenance would be viewed as corporate insanity. Yet this is what we are attempting here. Maintenance does not necessarily change functionality, but it may change a great deal. One type of maintenance is reverse engineering, which (by definition) should not change functionality, but rather re-documents requirements based upon the base lined operational system [4]. Function Point Analysis attempts to quantify functional changes in order to relate them to a percentage of design changes through code, testing etc. If the user requests functional changes, the same methods used to size new work apply with additional guidance that deleted functions add to the maintenance project (though they subtract from the baseline assessment).

Architecture and design analysis should be conducted to determine design changes. Correlation should be made to bi-directionally map between functional changes and design changes. Optimistically, a Requirements Trace Matrix (RTM) should be updated with each Change Directive (along with other base lined documents and specifications under configuration management.) This baseline update should provide accurate data for future change analysis sizing and pricing. Both application size and complexity impact corrective maintenance costs. How can one quickly and indisputably measure applications as impacting corrective maintenance costs?

Complexity

The term "complexity" needs to be treated very carefully. Often, all that is meant by complex is "poor structure"- i.e. the solution structure does not match well with the problem structure. Clearly, poor structure is the supplier's "fault" and it should be "factored out" in cross-project or cross-organisational comparisons as poor structure will make itself visible through poor productivity [5]. A well-structured solution is likely to cost less to fix and to enhance than a poorly structured one. However, all measurements must be used in the correct context to attain maximum usefulness.

Algorithmic complexity is an additional issue. The most important cost driver concerning algorithmically complex software lies in its verification and validation. When regression-testing techniques are correctly applied, this may be easier and less expensive than is the norm for business/commercial software. If documentation is of poor quality (an unfortunate norm) understanding the software may be time consuming [6]. Alternatively, complexity may be used as a synonym for "magnitude." Accounting for inter-relationships' complexity between data files and entity types is liable to lead to counting the same item twice.

The final complexity type is that which results from a mismatch between the application and the desired qualitative and performance requirements particularly with the operational platform (or the operational environment). For example, delivering a fault-tolerant application, serving thousands of users, with sub-second response time and 7x24 operations, is inherently more complex on a non-redundant platform, than delivering the same application using a platform designed for this. Not because there is anything necessarily wrong with this platform but because that is not what it was designed for. The correct way to deal with this complexity is to evaluate Critical Success Factors (CSF) for acceptable performance ranges, assess likely performance of the proposed solution and assess impacts and costs of each fix on each of the CSF. Application or suite complexity is not a controlling factor that matters, but rather complexity of the flaw being addressed by corrective maintenance. Development of mutual understanding between all people and organisations in the Acquisition loop (i.e., customers and suppliers) is a critical corporate-anthropological function.

The number, size and scope of maintenance corrections are actually functions of the quality of the development process (i.e., execution of each phase of the systems development life cycle). Impacts of each "flaw" on a running system are exponentially greater the earlier in the development cycle that they occurred.

Cost of Repair

Five factors combine to determine the cost of repair of a particular flaw:

A. Labour cost.

B. Size.

C. Complexity.

D. Range.

E. Domain.

Labour cost varies with the number of staff-hours and the required skill sets needed to repair a problem. To this must be added all of the expenses (both direct and indirect) of detection, identification, analysis and remediation and, of course, change management. To this one should add the business costs outside the system. These include reruns, data adjustments, client and user support costs, internal costs, etc. An outsourcing environment visually enhances all of these latter issues [7]. Once a problem has been detected, the size of the flaw must be analysed. Only then can the complexity of the flaw's repair be considered.

The range of the correction is determined by the quantity of applications affected by the flaw. The domain of the flaw refers to whether the correction affects other systems or business processes. What may first appear to be a minor computation flaw may develop into business process re-engineering, altering product lines carried by multiple distribution centres in a region with a multimillion dollar price tag result.

A Case Study

A good case in point is the description of the Patriot missile system, described by section 5.8, on pages 64 through 68 [3]. Mr. Stein said on page 59, "keep one's eye on the forest when dealing with details in the trees". At the end of a process of "Defense Systems Analysis", one must ask whether lives have been saved. Mr. Stein describes the process of upgrading the Patriot Missile system to deal with Scud missiles, from 1979, finally culminating in the PAC-II upgrade, just prior to the First Gulf War in 1991. Twelve years of effort. Unfortunately, Mr. Stein does not provide the end of this useful and interesting story. What was the result of this 12-year-upgrade-process? That, after all, must be the final test. DID THE SYSTEM FUNCTION AS PLANNED? Did it save lives?

Helmuth von Moltke the Elder, Germany's military leader in WW I, said, "no plan of operations extends with any certainty beyond the first contact with the main hostile force." I opine that a similar statement is applicable to weapons' systems, whether defensive or offensive. In the Gulf War, the Patriot system was installed in both Israel and In Saudi Arabia. (By the way, the system was "sold" to Israelis as an anti-aircraft system that had been specially modified for anti-missile, which Mr. Stein disagrees with.) During the War, 39 Scud missiles were fired at Israel. The Patriot system intercepted exactly zero incoming Scuds. In one case, there was estimated to be a ten percent chance that a missile's course was affected by the Patriot. Falling Patriot Missiles caused more economic damage than Scuds. (I was a senior Defense Systems Analyst at the time, specializing in quality and reliability.)

The system was a total failure, bordering on disaster; after 12 years of development to upgrade an existing system. This is rather beyond disappointment. Bottom line message: in Defence Systems Analysis, if you do not receive real data from the actual war situation, you still know nothing conclusive about your system. "War stories" MUST come from war. It is my belief that had this process been accompanied by a proper, professional anthropological study, this disaster could have been avoided.

References

1. Vogt, Edwards E (1995) The nature of work in 2010. Telecommunications, USA.

2. Bryen Stephen D (2015) Technology security and national power: winners and losers. Transaction Publishers, New Jersey, USA, p. 328.

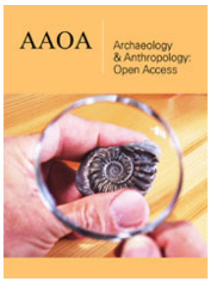
3. Trillium (1994) Model for telecom product development & support process capability. Bell Canada, Montreal, Canada.

4. Ben Menachem Mordechai (1994) Software configuration management guidebook. McGraw Hill.

5. Delaney William P (2015) Perspectives on defense systems analysis. MIT Lincoln Laboratory Series MIT Press, Massachusetts, USA.

6. Harper, Richard, Rodden T, Rogers Y, Sellen A (2008) Being human: human-computer interaction in the year 2020. Microsoft Research, p. 98.

7. Rzevski, George, Skobelev P (2014) Managing complexity. WIT Press, Ashurst, UK.


AAOA
Archaeology & Anthropology: Open Access