TITLE: "MANAGING DESIGN COMPLEXITY"

by Tim Bryce
Managing Director
M. Bryce & Associates (MBA)
P.O. Box 1637
Palm Harbor, FL 34682-1637
United States
Tel: 727/786-4567
E-Mail: timb001@attglobal.net
WWW: http://www.phmainstreet.com/mba/
Since 1971: *"Software for the finest computer - the Mind"*

"100% of your design documentation is contained in the specifications of your information resources"
- Bryce's Law

There are many companies today, most overseas, still tackling major systems projects particularly in the areas of banking and manufacturing. These mammoth application development efforts contrast sharply with American companies who have failed in such undertakings and are now content with chipping away at systems, program-by-program, with the hope that disjointed software will somehow/someday interface with each other. Whereas foreign competitors talk in terms of enormous systems with hundreds of programs and millions of lines of code; large integrated systems tend to intimidate the most ardent of American developers. But this is not so much a story about competition as it is about understanding design complexity.

People in both the east and the west recognize the design and development of a total system is no small task. A system can consist of many business processes, procedures, programs, inputs, outputs, files, records, data elements, etc. The problem lies in how to best control these information resources and the design decisions associated with them. Two approaches are typically used: progressively break the problem into smaller, more manageable pieces, or; tackle a minuscule portion of the problem at a time. Whereas the former requires a long term perspective, the latter can show a quick return, which is more appealing to a company with a "fast track" mentality.

Some time ago we conducted a study of customer application development projects. Our research centered on two types of projects: those aimed at building a total system, and; those aimed at building a single program. One obvious conclusion was that the number of information resources used in a major system was considerably more than in a program.

However, the key observation made in the study was that there is a finite number of design decisions associated with each type of information resource. As an example, for an output, decisions have to be made as to its physical media (screen or report), size (number of characters), messages associated with it, etc. For a data element, its logical and physical characteristics must be specified (definition, source, label, size, class, length, etc.). For a program, the language to be used, program logic, required file structures, etc. These design decisions can be simple or complex; regardless, they are all required in order to design a system or a program. When we multiply the number of design decisions by the number of information resources, we get an idea of the magnitude of a systems design project versus the design of a single program (see Figure 1).

From this perspective, the average system design project is nearly 25 times larger than the average software design project in terms of complexity. As a footnote, our findings also revealed the "average" system design project is seven times larger than a "complex" software design project.

This discrepancy in system/software complexity provides a clue as to how companies address the problem. Since a software design project is smaller and seemingly more palatable to implement than a total systems project, some companies will focus on software engineering tools and techniques, and abandon total systems engineering practices. This is one reason why programming tools enjoy popularity today.

Contrast this with the size of Japan's "Best" project to build the country's next generation of on-line banking systems. This was a major application development effort resulting in 72 "average" systems; a considerably larger project than what is typically addressed in the United States.

MANAGING DECISIONS

There are two aspects to handling decisions: how they are formulated, and how they are controlled.

Trying to make nearly 50,000 design decisions in one step is not only an impossible task, it is a highly impractical way of operating. Just like the design of any product, a system must be designed in gradual phases in such a way as it becomes possible to review and refine the design. In other words, the 50,000 design decisions

(coninued on page 2)

(continued from page 2)

will be made throughout the life of a development project, not all at once.

It is the responsibility of a systems engineering methodology to define the sequence of events for designing a system. As such, the methodology represents the channel for formulating decisions. Breaking a complex system design down into smaller, more manageable pieces, also provides for:

* Parallel development and delivery of portions of the system (concurrent development within a single project).

* An environment conducive for building quality into a product (as opposed to inspecting for quality afterwards).

* The formulation of Project Management related decisions (such as estimating and scheduling the delivery of systems, in part or in full).

This philosophy of design is no different than any other product design/development effort, such as shipbuilding, automobile manufacturing, bridge building, etc. All require a specific methodology that breaks the product down to its sub-assemblies and parts; thereby organizing the specification of parts and the design decisions associated with them.

Managing the decision making process for even the smallest of application development projects can be a huge undertaking. We estimate there are approximately 500 design decisions associated in a small software design project (as compared to more than 125,000 decisions in the typical complex system design project). To record and control these decisions requires something more sophisticated than just paper and pencil; it requires an automated "Information Resource Manager" (IRM), a software tool capable of inventorying and documenting

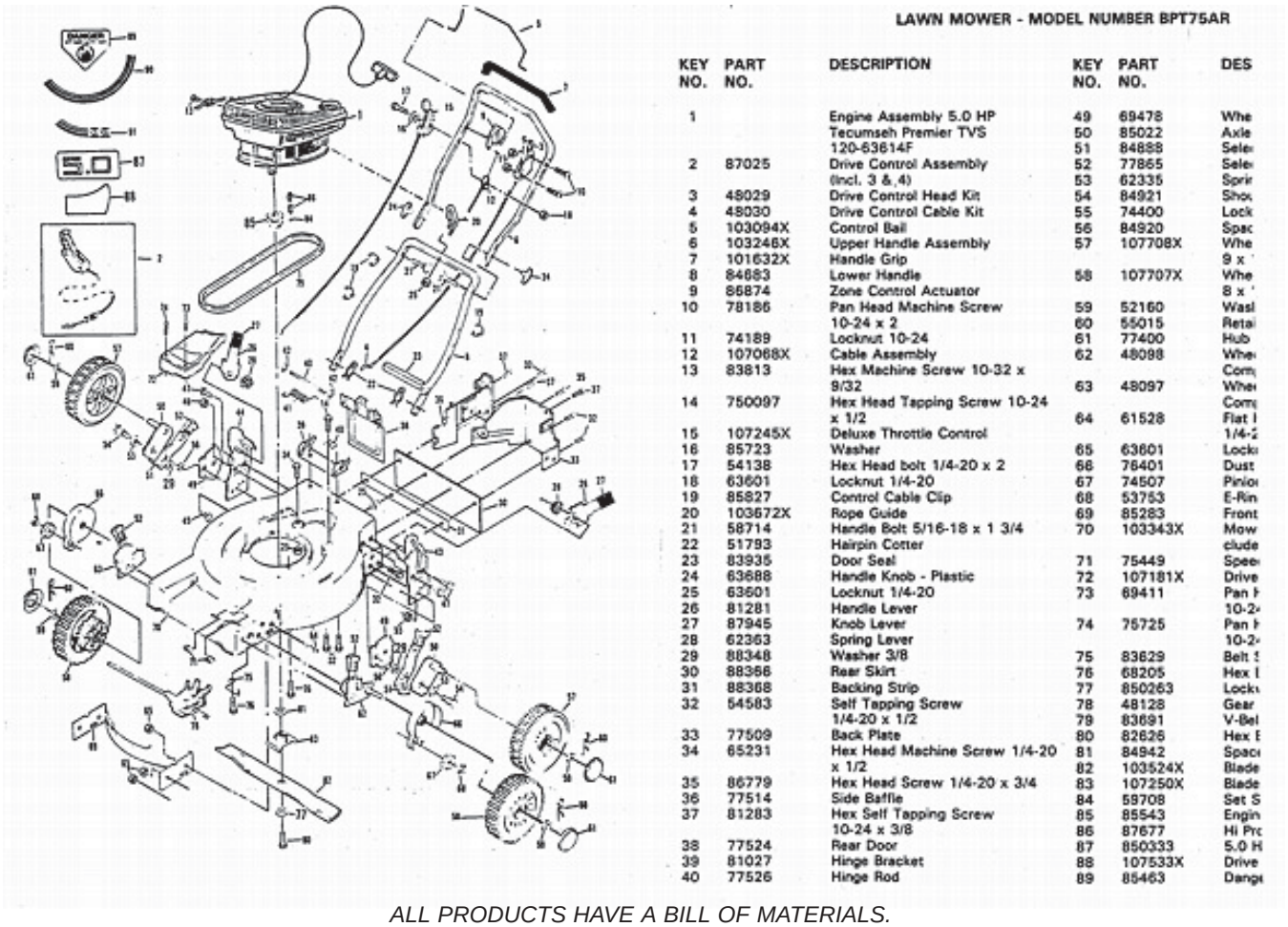
(continued on page 3)

FIGURE 1

	RESOURCES IN SYSTEM	RESOURCES IN PROGRAM	DECISIONS PER RESOURCE TYPE	DECISIONS IN SYSTEM	DECISIONS IN PROGRAM
SYSTEMS	1	0	25	25	0
SUB-SYSTEMS (business processes)	15	0	25	375	0
PROCEDURES (both Computer & Administrative)	40	0	30	1,200	0
PROGRAMS	75	1	30	2,250	30
OPERATIONS (manual steps)	125	0	10	1,250	0
INPUTS (interactive or batch)	50	3	15	750	45
OUTPUTS (screens & reports)	200	5	15	3,000	75
FILES (logical and physical files, computer and manual)	100	4	30	3,000	120
RECORDS (includes file structures, print maps, panels, input transactions, etc.)	1,000	10	30	30,000	300
DATA ELEMENTS	400	75	20	8,000	1,500
TOTAL NUMBER:	2,006	98		49,850	2,070

NOTE: Decisions are design oriented only; they do not include Project Management related decisions (such as those associated with planning, stimating and scheduling).

FIGURE 2 ***BILL OF MATERIALS CONCEPT***



LAWN MOWER - MODEL NUMBER BPT75AR

KEY NO.	PART NO.	DESCRIPTION	KEY NO.	PART NO.	DES
1		Engine Assembly 5.0 HP Tecumseh Premier TVS 120-63614F	49	69478	Whe
			50	85022	Axle
			51	84888	Sele
2	87025	Drive Control Assembly (Incl. 3 & 4)	52	77865	Sele
			53	62335	Sprir
3	48029	Drive Control Head Kit	54	84921	Shox
4	48030	Drive Control Cable Kit	55	74400	Lock
5	103094X	Control Rail	56	84920	Spac
6	103246X	Upper Handle Assembly	57	107708X	Whe
7	101632X	Handle Grip			9 x
8	84683	Lower Handle	58	107707X	Whe
9	86874	Zone Control Actuator			8 x
10	78186	Pan Head Machine Screw 10-24 x 2	59	52160	Wash
			60	55015	Retai
11	74189	Locknut 10-24	61	77400	Hub
12	107068X	Cable Assembly	62	48098	Whe
13	83813	Hex Machine Screw 10-32 x 9/32	63	48097	Com Whe
14	750097	Hex Head Tapping Screw 10-24 x 1/2	64	61528	Com Flat 1/4-2
15	107245X	Deluxe Throttle Control			
16	85723	Washer	65	63601	Lock
17	54138	Hex Head bolt 1/4-20 x 2	66	78401	Dust
18	63601	Locknut 1/4-20	67	74507	Pinio
19	85827	Control Cable Clip	68	53753	E-Rin
20	103672X	Rope Guide	69	85283	Front
21	58714	Handle Bolt 5/16-18 x 1 3/4	70	103343X	Mow
22	51793	Hairpin Cotter			clude
23	83935	Door Seal	71	75449	Spee
24	63688	Handle Knob - Plastic	72	107181X	Drive
25	63601	Locknut 1/4-20	73	69411	Pan H 10-2
26	81281	Handle Lever			
27	87945	Knob Lever	74	75725	Pan H 10-2
28	62363	Spring Lever			
29	88348	Washer 3/8	75	83629	Belt
30	88366	Rear Skirt	76	68205	Hex
31	88368	Backing Strip	77	850263	Lock
32	54583	Self Tapping Screw 1/4-20 x 1/2	78	48128	Gear
			79	83691	V-Bel
33	77509	Back Plate	80	82626	Hex
34	65231	Hex Head Machine Screw 1/4-20 x 1/2	81	84942	Spac
			82	103524X	Blade
35	86779	Hex Head Screw 1/4-20 x 3/4	83	107250X	Blade
36	77514	Side Baffle	84	59708	Set S
37	81283	Hex Self Tapping Screw 10-24 x 3/8	85	85543	Engin
			86	87677	Hi Pr
38	77524	Rear Door	87	850333	5.0 H
39	81027	Hinge Bracket	88	107533X	Drive
40	77526	Hinge Rod	89	85463	Dang

ALL PRODUCTS HAVE A BILL OF MATERIALS.
SO DO SYSTEMS & SOFTWARE.

(continued from page 2)

an enterprise's information resources.

Whether you call it an "IRM", a "Repository", a "Data Dictionary" or whatever, the philosophical heart of the product is based on the age-old concept of "Bill of Materials" whereby resources (also referred to as "components" or "parts") are cataloged and cross-referenced to each other. Consider a lawnmower product as shown in Figure 2. I am sure this type of diagram is familiar to any homeowner who has reviewed product maintenance/ warranty booklets.

In Figure 2, every part in the product is identified by number and name (see section to the right in the figure). To the left side in the figure is a schematic showing how each part relates to the other parts and, as such, represents the assembly of the product for maintenance purposes.

The concept of "Bill of Materials" provides the means to inventory resources thus allowing us to share and re-use them. For example, many of the parts shown in Figure 2 are re-used in other lawnmower models offered by the manufacturer. How can we share and re-use resources without such a concept? The answer is simple: we cannot. And this explains why there is considerable redundancy in our information resources and work effort. It also suggests most of our design decisions are maintained "by the seat of our pants." Most college courses involving computing are unfamiliar with the Bill of Materials concept. Their focus is on programming and file design, and little else.

The concept of "Bill of Materials" has three objectives:

1. To uniquely identify each resource by number and name (as well as by aliases). Names are nice, but numbers offer a more precise way to uniquely identify a

(continued on page 4)

(continued from page 3)

resource. Identification is critical. After all, we cannot share and re-use something if we do not know it exists.

2. To record the part's specifications. Thus providing a way to determine if the part can be re-used in another product (thereby promoting the sharing of parts and eliminating redundancy).

3. To record where the part is used in a product(s) (aka "Where-used"). This specifies the relationship of parts to each other and, thereby, their assembly. This is also extremely useful for "impact analysis" whereby we can analyze where the part is used in all of our products, not just one, which is vital for making intelligent decisions about modifying a part. For example, if we change the specifications of a part in one product, this will severely impact other products it is also used in.

By controlling parts in this manner, a product's design is fully documented.

The "Bill of Material" concept can easily accommodate information resources and offer the same benefits of sharing and re-using components. By doing so, we can easily manage the 50,000 design decisions accompanying a system design project. Our system/software products may be less tangible than an automobile, aircraft or lawnmower, but we can still apply the same concept to their control.

Therefore, an IRM Repository should have the ability to identify, specify, and cross-reference all of the resources mentioned in Figure 1. This can certainly be done manually with paper but this may lead to bureaucratic and access problems for developers. Instead, automation is recommended. There are several such commercial products on the market, but it is also fairly easy to create such software using today's Data Base Management Systems (DBMS) which are now fairly easy to define and relate resources (they also provide excellent documentation services).

The IRM should be viewed as the hub of all development efforts and provide the means to interface (import/export) with a myriad of other development tools; e.g., CASE, prototyping aids, program generators, etc. Such tools will use the intelligence of the information resources as contained in the IRM to function accordingly. As an example, a program generator should be able to interpret the program and file specifications in order to produce the necessary code. Such development tools should also have the ability to turn around and import resource specifications back into the IRM. This is particularly useful for documenting existing systems/software (aka "Reverse Population").

For information on how to create an IRM Repository, please see:

http://www.phmainstreet.com/mba/pride/spir.htm

The concept of "Bill of Materials" is an important part of an overall strategy to implement an "Information Factory" environment to design and develop information resources. But this will be the subject of a separate paper.

CONCLUSION

This philosophy to managing design complexity is no different than what is found in the engineering and manufacturing of any product. Engineers break their design projects into smaller stages so that reviews can be performed and revisions implemented. A "bill of materials" processor is used to track the parts or a product and how they interrelate; which is no different in intent than the IRM tool.

For people imbued in programming, it is difficult to think in terms of "parts" as described herein, but it is a practical solution and can be applied to any development effort, large or small. Standardization and integration of information resources is built by design, not by accident.

Without a formalized methodology for design or an IRM tool to record design decisions, a major system design is incomprehensible; there are just too many variables for the human mind to remember or control using manual techniques. It is not that analysts do not want to take on a major systems design project, they simply cannot. They lack the organization and proper tools to perform the job effectively. Because of this, they default to the things they know best, programming, and tackle systems in piecemeal.

The difference between east and west here is not one of working harder, but smarter. The Japanese and Europeans are simply better organized and equipped to perform system design than their American counterparts. This can be attributed, in large part, to management's sensitivity to the role systems play in a company. Because of this, they are not afraid to tackle large endeavors, while American companies view such undertakings as seemingly too massive to undertake. As such, they sidestep large projects in favor of smaller projects that may address only a portion of the overall problem. This

(continued on page 5)

(continued from page 4)

is resulting in the unsettling situation where our competitors are rapidly becoming the world's systems engineers, while Americans become the world's software engineers.

For more information on our philosophies of Information Resource Management (IRM), please see the "Introduction" section of "PRIDE" at:

http://www.phmainstreet.com/mba/pride/intro.htm#irm

END

"PRIDE" Special Subject Bulletins can be found at the "PRIDE Methodologies for IRM Discussion Group" at:

http://groups.yahoo.com/group/mbapride/

You are welcome to join this group if you are so inclined.

"PRIDE" is the registered trademark of M. Bryce & Associates (MBA) and can be found on the Internet at:

http://www.phmainstreet.com/mba/pride/pride.htm