AGLPM5 - Unit 2 - Activity 2: OBSERVE

Top Down and Waterfall Approaches

This document is an excerpt from the book:
"Agile Project Management for Government "
Authored by Brian Werham
Published by Maitland & Strong
Reproduced with permission under license

PMCAMPUS.com / Mokanova Inc
.

Chapter 19

The Lure of 'Big Design Up-Front'

Procrastination is the thief of time.

Mr. Micawber
from "David Copperfield"
by Charles Dickens

Big Design Up-Front (BDUF) and waterfall approaches go hand in hand. Where one is assumed, the other follows. There are institutional reasons why these approaches are often followed in government development projects:

- Top-down approaches are very appealing to hierarchical organizations
- Having detailed specifications appears to simplify procurement
- The sign-off and baselining of each progressive stage of a project appears to give certainty to the making of progress
- Advisors and consultants to large organizations may be motivated to tell the client what they want to hear, especially if it results in protracted strategy studies and analysis
- The use of approaches which produce a copious audit trail of documentation is appealing to bureaucratic organizations.

The Origins of BDUF

It is difficult to imagine today the influence that James Martin had over the development of IT practices in the 1980s. Bill Gates's Microsoft was still a sub-contractor to IBM, and Steve Jobs had left Apple and was spending his wilderness years trying to sell the ill-fated NeXt computer. The name on everybody's lips at the time was "James Martin" – the 'Guru of the Information Age'.

Martin had an amazing influence in selling the IT industry the waterfall model of planning, detailed specification, build and test. From his seminal 1983 book entitled "An Information Systems Manifesto", through to his huge 1,000 page, three volume work

"Information Engineering", he convinced many that the way forward was to carry out extensive strategic reviews and build comprehensive designs before starting any practical work – in other words he convinced many people that a Big Design Up-Front was a precondition for success.[1]

And now 30 years later, Martin has given away $150m of his fortune to his alma mater, Oxford University.[2] He lives in luxury on his private island in the Caribbean,[3] and is still writing books – adding to the over 100 that he has published over the last 50 years.[4]

In the late 1960s programming was maturing from a semi-amateur, part-time occupation into a profession in its own right. The programmers became more ambitious with their projects, not just writing payroll and accounting programs but stating to automate other business critical applications, such as stock control and invoicing. As the scope of the programs increased so did the size and complexity of the programs that they were writing. As they started working in larger teams the need for a method of coordination of their work became apparent.[5]

An approach to this organization was first suggested in a modest paper by H.D. Benington back in 1956 as a way of formalizing the development of software using a waterfall approach.[6] The resultant problems of using a waterfall approach have been picked over many times over the years.[7] Benington himself recognized the problem that operational testing (or 'shakedown' as he termed it) could take an indefinite time, and that total costs could never be predicted with accuracy.

Benington's paper was the seed of an emerging paradigm – early restrictions of the size of computers and lack of sophistication in tools forced a waterfall approach which continued to be adopted even as these performance restrictions fell away.[8]

In 1970 an influential Institute of Electrical and Electronics Engineers (IEEE) paper Winston Royce declared it to be "fundamentally sound". He advised that some additional features needed to be added to address most of the development risks:[9]

- Carry out early work to "assure that the software will not fail because of storage, timing and data (performance) reasons".
- Create a pilot model, test and use it about ⅓ of the way into the project to find "trouble spots in the design"
- Carefully plan, control and monitor testing which occurs "at the latest point in the schedule when ... alternatives are least available, if at all".
- Involve the customer so that "he has committed himself at earlier points before final delivery."

Despite Winston Royce's advice that an early prototype should be used, his emphasis was that the creation of 'quite a lot of documentation' was a priority, thus sign-posting the way for the move towards design work being seen as an output in its own right, above and beyond the production of working software.[10]

His recommendations should be seen in context. He was giving advice for programmers creating small, but mission-critical pieces of software written in difficult to read *machine code*. His expertise was in development of software for spacecraft leading up

to the successful 1969 mission to put a man on the moon – no mean accomplishment. However, his specific context, that of documenting small modules life-critical software as part of a complex mechanical control system, was soon lost, and was taken up as general advice for all software developers.

Programming on mainframes in those days was a slow and tedious process. Getting users to commit very early to a theoretical specification could save programmers a lot of time, as long as the specification was correct. This need to ensure correctness of mainframe specifications was a very strong influencing factor in the development of formal techniques in governmental organizations.

As the size and complexity of software development increased in the 1970s governments latched onto waterfall approaches with eagerness. They cast aside Winston Royce's insights into the value of prototyping and the importance of continual feedback. Standards such as DOD-STD-2167, SSADM and Merise were developed in the US, UK and France respectively. As discussed earlier in the discussion about the failures of the London Stock Exchange Taurus project (on page **Error! Bookmark not defined.**), SSADM and Merise focused on activities, not outcomes – teams were rewarded for creating volumes of beautiful, interconnected diagrams, not for creating solutions for users' problems. These methods concretized the ideas of popular works by Yourdon, Sarson and Gane and Tom DeMarco which were only intended as techniques for stimulating thinking, not as standards.

In the UK the SSADM method added Ted Codd's ground breaking mathematical ideas on data structures to create what became a rigid approach to requirements development.[11] A major study by Middleton found that:[12]

> "SSADM helps to give the appearance of administrative control over the complex process of software development … It has failed on two counts. First, it is based on a waterfall model of software development which is appropriate for only a small number of projects. Second, due to its flaws, complexity and lack of empirical base, it is not an effective way to raise skills, or direct the efforts of inexperienced software developers." [13]

By the mid-90s these methods had reached a high-tide mark of acceptance. Middleton carried out an in-depth investigation into 15 public sector organizations and several private sector companies found that the structured approach was neither suitable for large systems, nor for the development of small PC-based systems. None of the projects using this approach had delivered on time or to the users' needs. The standard defense of the proponents of the method argued that this was simply due to insufficient rigor in their use, and lack of tailoring of the method to the situation. However, again and again these approaches led to technical problems emerging very late in projects causing disruption and unexpected cost. After several years of work on these projects there were often no tangible results and the users had completely lost confidence in the process. In the case of small projects, the method was either disregarded, or tailored beyond recognition, and many had started to use prototyping or incremental development.

The assumption behind SSADM was that it would establish firm requirements at an early stage. But this pre-requisite is difficult to meet because users often do not know exactly what they need or what the technology can achieve. Changes to requirements were

continually needed as the users better understood the emerging solution and developers gained an understanding of the detail of the business and as external circumstances changed. The staff members who were being asked to use the method remained unconvinced until late in the process because the method was theoretical. Evidence that the method actually delivered benefits did not exist, and poor results seemed to suggest the opposite.

Using CASE Tools to Control the Problems of BDUF

Problems that the early users of these structured methods encountered were widely put down to two factors: one, the paper mountains of documentation that they generated and two, their tactical nature. The designs produced were criticized as only being specifications for individual software modules, without a grand view of the end-to-end system.

Two solutions to the problems of BDUF were proposed by James Martin. First, in association with Texas Instruments Corporation, he developed a Computer Aided Software Engineering (CASE) tool called the Information Engineering Facility' (IEF) to record the detailed documentation and ensure that cross-references were automatically updated. Second he created a strategic method called Information Engineering to encourage analysts to document a whole organization in complex diagrams and designs which would be typed into the IEF tool.

IEF fastidiously documented designs from strategy to detail, and then generated mainframe computer programs automatically from those details. Martin proclaimed that the end of the programmer was nigh!

Alongside the IEF, Martin's big-scale, top down Information Engineering method required a detailed waterfall of analysis and design activities. There were five steps to Martin's waterfall: Information Strategy Planning (ISP), then Business Area Analysis (BAA), then System Design (SD), then Construction and finally Cutover (together known as CC). The initial phases of ISP and BAA would typically take months to complete, after which more experts would be drafted in to spend months, and sometimes years on SD. Many customers gave up at this stage, not having seen any practical output from this work. Others, who persevered, found that the resultant systems that were generated at the end of the process would only work on mainframes, not on the more efficient and flexible personal computers and mini-servers that were becoming available.[14]

The high point of CASE tools was in 1990. As IT departments remained fixated on use of mainframes, end-users became increasingly frustrated and started to buy and program PCs and mini-computers of their own. This revolution in *end-user computing* started to eat into IBM's revenues, so it attempted a last ditch attempt to lock customers into the dying mainframe technology. This new method was called AD/Cycle, and was the last significant attempt to create a waterfall structured method with an associated complex and expensive toolset. AD/Cycle was abandoned after just two years in 1992 and signaled the death of the Information Engineering approach.[15]

Although he had belatedly become interested in iterative development, Martin's voluminous writings on Rapid Application Development (RAD) only contained a few pages

on the topics of prototyping and timeboxing which are vital for iterative development.[16] In fact, worries about the usefulness of outputs from RAD developments were a prime motivator for the development of DSDM.[17]

In an influential paper, Beath and Orlowski noted that Information Engineering was ideological in nature, not evidence-based. They accused Martin of using a patronizing tone towards users who they say were "portrayed as naïve, technically unsophisticated and parochial". They note that the amount of action expected of users in the analysis stages was actually only about 2%, far below the level of user participation and collaboration promised by the method.[18]

Many empirical studies conducted found either very little or no productivity improvement from the IE code generation process. Some research at the Ford Motor Company in the UK showed an 85% improvement in output when they switched from using SSADM paper specifications to using Information Engineering supported by the automated IEF tool. However, it is not clear whether this was just due to the dumping of the old method rather than the adoption of the new method and tools. It was certainly much less than the 300% improvement that Ford was promised. It has not been publically disclosed whether the $9m exercise plus additional on-going training, expense of skills acquisition and recurring license fees gave a positive return on investment or not.

Usage of the IE method and the IEF tool began to wane because feedback from the users involved in these projects was poor. IT departments also started to baulk at the incredibly high recurring license fees for the IEF tool together with its very restrictive licensing and dependency on costly mainframe technology.[19] Various attempts to revive interest in the Information Engineering approach were made, but most companies discovered that it was quicker and cheaper to program directly into the mainframe rather than draw all the diagrams first.[20]

Using Better Techniques to Solve the Problems of BDUF

With interest in IEF fast fading, Martin turned to another form of BDUF called Object Oriented Analysis and Design (OOAD).[21] In structured analysis, documentation was produced in two volumes: a data definition and a process definition. In writing a program a developer would have to constantly cross-reference between the two. OOAD was an attempt to integrate these processes and data analyses under one documentation standard. As with the previous structured methods OOAD was initially proposed as a modest, user oriented, approach,[22] but its use was developed by proponents into grander and grander schemes until a new BDUF approach was born called the Unified Modeling Language (UML).[23]

Research shows that UML has a clear benefit where the functional correctness of small but complex systems is paramount, which confirms Bennington's original thesis. For checking the internal logic of these systems it has its uses, but most practitioners now believe that its use should be limited. There is a significant learning curve to its adoption, and no time savings in development have been found in the research into its use. Indeed, for simpler tasks the researchers on one study found that "the time needed to update UML

documentation may be substantial compared with the potential benefits"[24]

Barry Boehm is a critic of the continuing use of complex modeling. He has stated that "most published object-oriented analysis and design (OOA&D) methods inadequately address the critical aspects of system performance".[25]

A three-year experimental study agreed that the benefits of UML were marginal at best, but concluded with the hopeful statement that:

> "The potential benefits of UML in the mid and long term are probably larger than what was observed in this experiment." [26]

Conclusions

The rise and subsequent fall of the most extreme waterfall approaches occurred from mid-1960s through to the early 1990s. The growth of the use of PCs during the end-user computing revolution, and what is now common usage of complex systems by everyday folk over the Internet has meant that non-IT professionals have appropriated control of areas of IS development that once were the sole preserve of the IS elite. If BDUF is a busted flush, what then for waterfall project management? What we have in this second decade of the 2000s is an *inflexion point*. We have reached a situation where we cannot continue with waterfall project management for technology developments.

Questions

1. Identify problems that you have seen in the past when a top-down approach to designing a solution is taken? Has it resulted in BDUF? Was a working solution produced?

2. In your current organization, are there certain processes where requirements are agreed in detail up-front before development of a solution commences?

3. Does any project you are currently involved in have elements of a waterfall approach? If BDUF is a pre-cursor for waterfall, how could your organization reduce these risks?

4. Read Middleton's critique of SSADM and its proposed adoption as a European standard (see Endnote 27). What are the main objections to SSADM that he cites?

5. Read Beath and Orlowski's entertaining critique of Information Engineering (see Endnote 28). What features of IE do you think helped to fuel the end-user computing revolution?

[1] Agile was not the first movement to use a manifesto to provoke a religious like zeal to its converts! See {Martin 1983 #60} and {Martin 1989-90 #61}
[2] $100m as a donation in 2005, and an additional $100m was raised in 2010 with James Martin providing half as matched funding alongside other donors such as George Soros {Futurist pledges $50m in matched 18/01/2012 #59}
[3] {The $100m man 18/01/2012 #57}
[4] {James Martin 18/01/2012 #58}
[5] To see the debate unfolding see examples such as {Naur 1969 #51}
[6] {H.D. Benington 1987 #47}
[7] See {Leffingwell 2007 #48: 17} and also {Larman 2006 #52}
[8] Benington experiences were with hand machine coded programs running on IBM SAGE computers with only 65k of memory. It is easy to read too much into his paper which was limited to "programming problems that are likely to arise during Forrester's 1960-1965 period of real-time control applications".
[9] {Winston W Royce 1970 #50: 2}
[10] {Winston W Royce 1970 #50: 332}
[11] {Kimble 13/10/2008 #54} and read about Ted's concepts here {Stonebraker 1988 #432}
[12] {Peter Middleton 1994 #65}
[13] {Peter Middleton 1994 #65}
[14] {Martin 1989-90 #61}
[15] {Mercurio et al 1990 #56}
[16] In fact only 29 pages out of 788 refer to prototyping, timeboxing and iterative development {Martin 1991 #68: 216-227, 312, 172-188}. This mammoth book was supported by 6 hours of video tapes emphasizing upfront analysis, planning and design before coding. See {Martin 1991 #68: 351} and also the complex intricate diagrams used in ISP {Martin 1991 #68: Figure 21.9}.
[17] {Craddock 2012 #330: 2}
[18] {Beath 1994 #70: 372} and {Beath 1994 #70: 361}
[19] {Finlay 1994 #69}
[20] {Martin 1985 #66}
[21] {Martin 1992 #67}
[22] {Coad 1990 #72}
[23] {Rumbaugh 1999 #74}
[24] {Arisholm 2006 #75}
[25] {Boehm 2001 #198: 8–9}. Boehm notes that *in a recent survey of 16 OOA&D books, only six listed the word "performance" in their index, and only two listed "cost."*
[26] {Dzidek 2008 #76: 17}
[27] {Middleton 1994 #65}
[28] {Beath 1994 #70}