Reducing System Software Project Risk through Choice of Project Architecture

by

John E. Arwe

B.A., Computer Science (1986)
Rutgers College

Submitted to the System Design and Management Program in Partial Fulfillment of the Requirements for the Degree of Master of Science in Engineering and Management at the Massachusetts Institute of Technology

June 1999

© 1999 John E. Arwe. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part.

Signature of Author .......................................................... System Design and Management Program

June, 1999

Certified by .................................................................

Michael A. Cusumano
Sloan Distinguished Professor of Management
Thesis Supervisor

Accepted by .................................................................

John R. Williams
Associate Professor of Civil and Environmental Engineering
Co-Director, System Design and Management Program

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
APR 12 1999
ARCHIVES
LIBRARIES
Reducing System Software Project Risk through Choice of Project Architecture

by

John E. Arwe

Submitted to the System Design and Management Program in Partial Fulfillment of the Requirements for the Degree of Master of Science in Engineering and Management

ABSTRACT

The choice of project architecture – the structure of and interrelationships between product, processes, and organization – alters the project’s risk profile. While most analyses take project scope as an input, I propose the examination of multiple project decompositions take place as part of project planning and project monitoring. The sub-projects created by each decomposition will have unique risk profiles, suggesting different process and organizational adaptations that lower overall project risk. By selecting project decompositions that partition risk and then adapting the structure of each sub-project to mitigate its particular risks, the probability of risk occurrence is reduced and the severity of consequences may be reduced. Case studies of four IBM mainframe system software projects illustrate lessons regarding project architecture, some general and some project- or process-specific. These projects employ both waterfall and iterative process models, managed using varying degrees of functional, lightweight, and heavyweight organizations.

Thesis Supervisor: Michael A. Cusumano
Title: Sloan Distinguished Professor of Management
# Table of Contents

INTRODUCTION ...................................................................................... 7

- How Should Projects Be Structured? .............................................. 7
- How Can the Need for Project Restructuring Be Recognized? .... 9

LITERATURE REVIEW ........................................................................ 9

A Fit-Based Project Architecture Approach ................................ 11

RISK ASSESSMENT ............................................................................. 14

DECOMPOSITION ............................................................................... 16

ADAPTATION AND FIT ASSESSMENT .............................................. 16

CONTEXTUAL BACKGROUND ............................................................ 18

- System, Middleware, and Application Products ....................... 18
- Product History ........................................................................ 19
- Process History ........................................................................ 21
  - Release Content Planning .................................................... 25
  - Organization History .......................................................... 25
    - Incentives ......................................................................... 29
    - Conflict Resolution ......................................................... 30

CASE STUDY OVERVIEW ................................................................. 31

- Major Risk Categories ............................................................... 32
- Data Collection Methods ........................................................... 33

CASE STUDY: WORKING SET MANAGEMENT ............................. 34

- Background ............................................................................ 34
- Stakeholders .......................................................................... 34
- Primary Risks Addressed by Project Plan ................................ 35
- Product Plan .......................................................................... 36
- Process Plan .......................................................................... 36
- Organization Plan .................................................................. 37
- Project Evolution .................................................................... 38
- Discussion ............................................................................. 40
- Lessons .................................................................................. 41

CASE STUDY: WORKLOAD MANAGER RELEASE 1 .................... 44

- Background ............................................................................ 44
- Stakeholders .......................................................................... 44
- Primary Risks Addressed by Project Plan ................................ 45
- Product Plan .......................................................................... 46
- Process Plan .......................................................................... 47
- Organization Plan .................................................................. 48
- Project Evolution .................................................................... 50
- Discussion ............................................................................. 52
- Lessons .................................................................................. 53

CASE STUDY: WORKLOAD MANAGER RELEASE 4 .................... 55

- Background ............................................................................ 55
- Stakeholders .......................................................................... 56
- Primary Risks Addressed by Project Plan ................................ 57
- Product Plan .......................................................................... 57
- Process Plan .......................................................................... 58
- Organization Plan .................................................................. 58
- Project Evolution .................................................................... 59
CASE STUDY: COMPONENT BROKER FOR S/390 .................................................. 63
  BACKGROUND .................................................................................................. 63
  STAKEHOLDERS ................................................................................................. 64
  PRIMARY RISKS ADDRESSED BY PROJECT PLAN ............................................ 65
  PRODUCT PLAN .................................................................................................. 66
  PROCESS PLAN .................................................................................................. 66
  ORGANIZATION PLAN ....................................................................................... 67
  PROJECT EVOLUTION ......................................................................................... 69
  DISCUSSION ........................................................................................................ 70
  LESSONS ............................................................................................................. 72

APPLYING THE SUGGESTED APPROACH TO S/390 SOFTWARE ....................... 75

  ADVANTAGES OF THE FIT-BASED PROJECT ARCHITECTURE APPROACH .... 76
    Both Project Structure and Change Timing Issues are Addressed .................. 76
    Low Implementation Cost .............................................................................. 76
    Project-Wide Scope is Encouraged ................................................................. 76
    Adaptation is Encouraged ............................................................................ 77
    Process, Organizational, and Environmental Aspects are Highlighted ........... 77

RETROSPECTIVE: S/390 CASE COMPARISONS BY PROJECT AND BY SUB-PROJECT 77

CONCLUSIONS ..................................................................................................... 80

BIBLIOGRAPHY ................................................................................................. 83
List of Figures and Tables

Figure 1: Case study time line ............................................................................................................. 7
Figure 2: Contrasting approaches to project architecture selection .................................................. 8
Figure 3: Risk prioritization example .................................................................................................. 15
Figure 4: Risk mitigation approach as a function of severity of consequences, probability of occurrence, and mitigation cost ........................................................................................................ 17
Figure 5: Simplified Mainframe Software Environment ...................................................................... 20
Figure 6: OS/390 and MVS Structural Overview .................................................................................. 21
Figure 7: Modified waterfall process overview .................................................................................... 22
Figure 8: S/390 Formal organization overview ..................................................................................... 28
Figure 9: Summary of requirements and planning process ..................................................................... 29
Figure 10: Working Set Manager Schedule Overview ............................................................................. 37
Figure 11: Working Set Manager Organization Overview ....................................................................... 38
Figure 12: Iteration process flow ........................................................................................................... 39
Figure 13: Working Set Manager schedule overview, showing iterations ............................................... 40
Figure 14: Workload Manager Release 1 Schedule Overview ................................................................. 48
Figure 15: Workload Manager Release 1 Organization Overview, after reorganization ....................... 49
Figure 16: Workload Manager Release 4 Schedule Overview ................................................................. 58
Figure 17: Workload Manager Release 4 Organization Overview ............................................................ 59
Figure 18: Early Component Broker/390 Schedule .............................................................................. 67
Figure 19: Component Broker/390 Functional Organization Overview ................................................ 68
Figure 20: Component Broker/390 Cross-Functional Team Overview .................................................. 69
Figure 21: Revised Component Broker/390 Schedule .......................................................................... 70
Figure 22: Summary of Case Project Decompositions, Adaptations, Risks, and Fit ................................. 79

Table 1: Mapping of waterfall process stages to IBM programming process architecture ..................... 23
Table 2: Process Step Definitions .......................................................................................................... 23
Table 3: Mapping Organizational Units to Process Steps ........................................................................ 27
Table 4: Case Study Comparison ........................................................................................................... 31
Table 5: Workload Manager Release 4 Functional Summary ................................................................. 55
Acknowledgements

I wish to thank the following people, who helped me on this long road.

Michael Cusumano, my MIT thesis advisor, who helped me organize my thinking and present it coherently in between conference calls with the Department of Justice, trips to other continents, and getting his next book to the publisher. It was great working with you.

Carroll Fulkerson, my IBM thesis advisor and former SDM inmate, who somehow found the time to provide input, relate project history, read drafts, provide comments, and have more conference calls than I care to remember over the past few months. How you did this in the midst of changing jobs, moving 600 miles, and increasing the size of the Fulkerson dynasty, I will never know.

Ted Mauerer, a more recent SDM inmate, who provided some necessary balance on the CB/390 side. This was especially important since I had no personal experience on that project, unlike the others.

Susan Puglia and Tom Rosamilia, my former and current VPs, for encouraging me to take on this topic and providing me with access to an in-flight project. Art Burchfield, CB/390 project manager, for not getting too upset when I started digging on a project you were just taking over even as customer deadlines approached.

Greg Lacey, my manager, for dealing with the lawyers and (along with Peter Yocom) for allowing me to be in Cambridge for the entire full-time SDM program (and then some).

And a cast of many other colleagues, names withheld, for searching through files and synapses in order to provide information used here. Although I’ve never met some of you other than via email, the data and perceptions you provided helped shape this work. I thank you all.

Trademarks

The following are trademarks or registered trademarks of the International Business Machines Corporation in the United States.

<table>
<thead>
<tr>
<th>IBM®</th>
<th>S/390®</th>
<th>S/370™</th>
<th>MVS®</th>
<th>OS/390®</th>
<th>RACF®</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMF™</td>
<td>SOM®</td>
<td>AIX®</td>
<td>OS/2®</td>
<td>CICS®</td>
<td></td>
</tr>
</tbody>
</table>

The following are or may be trademarks or registered trademarks of their respective companies.

<table>
<thead>
<tr>
<th>Unix®</th>
<th>CORBA</th>
<th>OMG</th>
<th>IIOP</th>
</tr>
</thead>
</table>
Introduction

This thesis examines issues related to the development of complex software. Four case studies of operating system projects are drawn from the System 390 (S/390) division of IBM. The first three cases comprise three stages in the evolution of a particular operating system component, the most recent of which was completed in 1997. The fourth case examines a new platform-specific product implementation project, in progress at the time of writing. These cases are examined in terms of the interaction between project risk factors, product characteristics, process capabilities, and the organizational environment.

![Case study timeline](image)

**Figure 1: Case study timeline**

The thesis goal is to develop a pragmatic procedure, usable in practice by software project managers, to answer two simple questions:

1. How should projects be structured?
2. How can the need for project restructuring be recognized?

The context within which these questions are posed is one of risk: the organization under study recognizes that certain types of projects are problem-prone, and is heavily risk averse.

The remainder of this chapter introduces the conceptual model used throughout, a general approach to architecting projects in order to mitigate risks. Later chapters provide detailed background on the IBM S/390 software organization’s environment and history; a short overview of the four cases studied, including a high level comparison of their major attributes; the four case studies; the general approach applied in the specific context of IBM’s S/390 software organization; and finally, major lessons generalized from the cases.

**How Should Projects be Structured?**

The assertion here is that the degree of fit between a project’s structural elements – product, process, organization, and environment – mediates the effects of risk factors, either amplifying or dampening them. Increased fit should reduce the probability of risk occurrence, and in some cases reduce severity of consequences should the risk be realized. Thus, this is a contingency rather than a congruence theory: “Central to a structural contingency theory is the proposition that the structure and process of an organization must fit its context (characteristics of the organization’s culture, environment, technology, size, or task), if it is to survive or be effective” [Drazin1985a]. Two kinds of fit are described in the literature: internal and environmental (or external) fit. Internal fit is defined by [Mile1992a] as “complementarities among aspects of structure and process”, while “Environmental fit demands that organizations match their
structures and processes to their external settings.” As used here, internal fit describes the degree of complementarities between the project’s elements, whereas external fit compares the project to everything outside of the project boundary, whether internal or external to the project’s company.

The process being operationalized here is achievement of optimal (maximum) overall fit between a project’s structural elements (internal fit), subject to the organization’s constraints. In other words, how to architect the project itself rather than the artifact(s) under construction. By using the project architecture to reduce risk, one gains significant leverage from the downstream effects. As with defects, the earlier in the process of delivering a project that risks can be removed (ideally, avoided), the greater the expected savings [Boehm1984a]. A physical analogy can be drawn by imagining the project is a pipe, through which resources (time, people) flow to accomplish work. If gaps exist between the project’s pieces, the pipe leaks: resources are lost, and less work is accomplished than expected given the inputs. Because the choice of pieces is influenced by the environment and by the work to be done, some gaps are likely but it is still desirable to minimize gaps so the pieces fit well together. Depending upon the material flowing through the pipe, gaps of a certain size may or may not result in measurable losses.

The transition from a risk to fit orientation has some interesting effects on project architecture selection. In both risk and fit views, the initial selection steps are risk identification and prioritization, but later steps diverge as shown in the figure below. An approach based purely on risk management leads to selection of mitigation actions for each risk, inherently a bottom-up process. An approach based on fit, however, combines bottom-up risk information with a top-down adaptation process: finding “the ideal” pieces for a project, contrasting it to the planned one(s), and modifying the product, process, organization, or environment to move closer to the project’s ideal until risk is reduced to an acceptable level. Significant conflicts between a project’s internal elements warns of a project granularity problem: it might be better to decompose the project into distinct sub-projects before selecting a process and organization structure, as happened in several of the case studies, or change the product content in a way that removes the conflicts.

![Figure 2: Contrasting approaches to project architecture selection](image_url)
How Can the Need for Project Restructuring be Recognized?

The monitoring aspects of project management are used to detect problems, but these mechanisms are not perfect. Even once lagging indicators show that problems exist, optimistic expectations of future progress are common. The landscape is littered with late and/or over-budget projects of all types [McFarl1981a, Rothf1988a, Neuma1988a, Oz1994a, Saltz1998a]; these projects did not just fail without warning. It is not clear how to distinguish between project management, technical, and structural problems [see McFarl1981a for a similar breakdown, which replaces project management with project size]. Project management problems are those arising from lack of monitoring metrics or failing to address problems detected by the monitoring system. These problems are generally found in projects described as “out of control” by their management. Project management problems represent a failure to adequately control risk. Technical problems are those arising from difficulties inventing or exploiting technology, often technology that is new to the organization in which the project is situated. Technical problems represent the realization of downside risk potential. Structural problems are those which arise from inconsistencies between the project’s structural elements – product, process, organization, and environment. This disparity (lack of fit) can be caused by any of several factors, including: a faulty project selection process, a working project selection process which is overridden due to business needs, and a poor choice of project decomposition.

Literature Review

Risk management in software is hardly a new idea [Boehm1989a]. Product risks are well documented, and there exists a substantial body of research on the generic risks of product development. Software products probably vary from others in the weighting of various risk factors due to its knowledge-intensive nature, but it is not clear that major risk factors specific to software exist. Some authors present methods for analyzing risks using isorisk curves, probability, and decision theory [Boehm1989 sections 1 and 2, AirFo1988a]. [Davis1988a] introduces an interesting graphical method to visualize the differences between software process models, but it is limited by the relatively small number of vectors along which graphical comparisons can be made. In this framework, risk loosely corresponds to the area between the customer needs and process vectors. [Aoyam1998a] extends the hardware concept of agility to the software development process, in effect assuming that development cycle time is the most important process risk.

Process concerns in software are supported by a body of knowledge dating back to the 1950’s [Brook1975a]. Boehm’s spiral model of software development uses risk as the basis for choosing process and content [Boehm1988a, Wolfl1989a, Boehm1994a]. [Alexa1991a] presents a methodology for selecting process models based on a numerical fit generated by comparing a set of twenty project attributes against the strengths of each process model. In this congruence model, each candidate pairing of project and candidate process model generates a fit value; the model showing the highest value is selected. Timing of market delivery (i.e. staging) is a level of process model selection independent of the development process. The level of attribute detail is similar in spirit to that suggested here (high/medium/low), although the form is somewhat different.
Likewise, the twenty selection criteria correspond roughly to the risk factors here, using five categories rather than four. All criteria are assumed equally important in this model.

[Perez1996a] and [Perez1994a] also use a congruence model, generating a congruence index to quantify the fit between process and context. This research is based on studies of 15 projects in the business application sector, concentrating on the requirements process. The results of adaptation were studied across 6 of the 15 projects, using qualitative measures translated into quantitative congruence indices. Three classes of adaptation were described:

1. Adaptation of a generic process to different contexts (environments), where adaptation resulted in a better fit (higher congruence index) in all cases.

2. Adaptation of the context to the prescribed process, where no process changes were implemented but the congruence index was already high.

3. Cases where the process and context were already “considered to be congruent”, so only minor adaptations occurred which increased the congruence index slightly.

Two important issues were noted by [Perez1996a] regarding quantitative fit metrics: perfect fit is an unreasonable expectation, and above a threshold (.6 to .7 using their values), context and process could be considered to be effectively congruent. These values appeared to be the maximum achievable using incremental process changes.

Organizational risk factors in software have a relatively small body of research, although there is an abundance of literature documenting experiences, admonitions, and war stories [Hackm1990a]. There is, of course, a large body of group theory research on teams [Thamh1987a, Brown1995a, Ancon1992a], some including software teams [Katz1997a, Curti1988a, Cusum1997b]. [Gruhn1992a] touches on the organizational aspects of software processes; while the term risk is not specifically used, the context is one of project success and failure. [Curti1988a] presents a layered organizational behavior model based on field study data concentrating on the soft factors of large software system design, including such issues as the decomposition of large design teams into smaller coalitions, and the resulting effects on quality and productivity. The integrative model of product development advanced in [Brown1995a] draws on the various streams of product development studies to understand how organization affects process and product:

"...we argue that (a) the project team, leader, senior management, and suppliers affect *process performance* (i.e., speed and productivity of product development), (b) the project leader, customers, and senior management affect *product effectiveness* (i.e., the fit of the product with firm competencies and market needs), and (c) the combination of an efficient process, effective product, and munificent market shapes the *financial success* of the product (i.e., revenue, profitability, and market share)."
Environmental risks, like product risks, are given treatment in the general project management and group theory domains [Ancon1992b, Shtub1994a]. Group theory contributions include behaviors such as gatekeeping and ambassadorial functions, whereby the external environment is altered to enhance project performance. This can occur through varied means, such as making new resources available or clarifying the performance objectives that will be used to gauge success by stakeholders.

Most authors neglect the choice of project architecture as a way to reduce risk, as well as interactions amongst the elements. Some cleave projects along a fixed dimension such as product structure [Cusum1997a, Cusum1997b] and then organize to match that division. Decomposition into sub-projects to improve the alignment between project and environment is not often addressed – the project scope is usually an input. [vonHi1990a] concentrates on the allocation of particular tasks in innovation projects by using quality function deployment (QFD) to reduce intertask problem-solving interdependence and boundary-crossing overheads. Von Hippel specifically mentions the product design to process design and marketing research to product design interfaces (which are determined by project architecture), asserting that partitioning can be performed as the project unfolds without increasing risk [vonHi1990a]. By and large, the literature assumes that either the elements can be tailored so that all three major risk categories are addressed (a project-oriented culture), or that upstream project selection will only choose projects which match the organizational and process capabilities (including acquisition of skills if necessary).

**A Fit-Based Project Architecture Approach**

The following sequence of steps assumes that the initial decomposition is done by product functionality, with process and organization being chosen based on this structure. If other cleavage planes are used to decompose the project, the independent and dependent variables in this approach must be updated accordingly.

1. **Identify risks.**

2. **Consolidate risks into major categories:** product, process, organizational, and environmental. **Assess probability of occurrence and severity of consequences** at the major category level, using a qualitative high/medium/low rating system. The use of an unfamiliar programming language would be classified as a process risk, for example, if infrastructure for the new language was not already in place. Requirements instability is a product risk; a necessity to work closely with a distant part of the organization (especially if local cultures differ) would be included under organizational risks.

Typically, environmental risks will be interproject rather than intra-project, requiring ambassadorial/gatekeeping behaviors or re-scoping of the project to mitigate. In this sense, environmental risks are largely independent of project architecture. For instance, if maintaining cooperation with outside groups having different agendas (e.g. a coalition of divisions or companies) is necessary for project success, specific personnel can be assigned liaison duties to influence and manage communication
with those groups.

3. **Assess project fit**, and if the project risk profile fits the proposed process and organization, stop. Most organizations have a small number of preferred processes and organizational styles; each of these can be considered. Using the preferred forms for these elements should generate good external fit, and if internal fit is also good then no further analysis is necessary.

For an organization such as S/390, functional organization and a modified waterfall process are preferred so the project would be evaluated with this combination first. Projects with highly uncertain requirements, which do not fit well with a waterfall process, should be evaluated with a number of process models to see if iteration or evolutionary development would provide a better fit.

4. **Generate candidate project architectures** whose sub-projects partition risk and whose elements can be chosen to match those risks. An ideal architecture partitions intraproject risk so that two of the three categories have low risk; an acceptable one has only one high intraproject risk category. Decomposition along different cleavage planes combined with different adaptations should be attempted if necessary to achieve this goal.

a. **Decompose into sub-projects** that have fundamentally different intraproject risk profiles. The different risk profiles provide opportunities to use processes and organizational structures best equipped to mitigate each sub-project’s risks.

In designing a VCR remote control, the control pad layout and the power system are so different that no one thinks twice about designing them independently, using different processes. The control pad layout requires feedback from users, and hence is often prototyped, while the power system is primarily constrained by electromechanical characteristics and target market availability of replacement power cells. There is every reason to apply the same technique to the design of diverse system elements such as user interfaces and internal algorithms.

b. Having established one or more decompositions which partition risk effectively, **identify a target organizational structure and process** for each element under each candidate decomposition. The targets should be “theoretical bests”; target is used rather than best to emphasize the qualitative nature of this step. As an example, a project requiring heavy coordination of personnel from many parts of the organization will have less communication overhead if the personnel report to the project and are co-located (heavyweight project management organization).
c. \textit{Generate possible adaptations to achieve better internal fit}, and assess the costs of each. Any of the project elements, including product content/scope, may be altered in an adaptation. Adaptations in one element (e.g. process) may cause co-requisite adaptations in other elements (e.g. organization).

Adaptations include large-scale project changes, such as choosing an iterative process for a sub-project with high uncertainty, or smaller changes such as adding code reviews in an application with higher-than-normal reliability requirements. A decision might also be made to change product scope by deferring high-risk or poorly understood sections of the product to a later release.

5. \textit{Select a decomposition} and corresponding set of actions from those generated above. The most important criterion is internal fit; complimentarity of the project’s product, process, and organization should be maximized. The match between product and process is particularly important, since they are likely to reinforce each other. If options with equally good internal fit are found, external fit is the tiebreaker.

By maximizing internal (intraproject) fit, it is normal to reduce external fit, which in turn requires more careful external stakeholder relations management. A project element with high requirements or innovation uncertainty might be developed using an iterative prototype process. While this satisfies the project’s need for rapid learning (good internal fit), it is common for the prototype team to become isolated from its surroundings (reduced external fit). Both sides conclude, “they just don’t understand.” Those outside the team see their peers/subordinates developing new norms that conflict with established modes of behavior. “Teams feel that they perform well when they concentrate their efforts internally; they reveal perceptions of performance that are negatively related to the frequency of [ed.: external] communication…” [Ancon1992b].

If no decomposition can be found that partitions risk effectively, it is best to buy information that will alter the risk profile before proceeding. This can be done in any number of ways, including limited-scope prototypes of areas with high uncertainty, use of external consultants, or simply waiting while current experts increase their familiarity with the problem domain.

6. \textit{Alter product, process, and/or organizational parameters} to achieve a better fit with the target selected above. In other words, implement the actions selected in the previous step.

This approach’s answer to the first question posed (How should projects be structured?) is that projects should be structured so that risk categories are partitioned amongst its elements. This allows for customization of the elements’ content within each partition to be chosen to match its major risk, in other words increasing the internal fit within each sub-project. The second question (How can the need for project restructuring be recognized?), as stated, becomes trivial: one can assess the fit at any point in time, and make whatever changes are deemed necessary to improve the fit. This result is not very
helpful to practitioners, however. What underlies this question is timing: when should projects be reassessed, so that needed changes can be made. Recent work in organizational theory suggests that the project's temporal midpoint is the most effective time to make such changes [Gersi1988a, Gersi1989a]. At the midpoint, time-limited groups are likely to accept change, and to solicit external stakeholders’ input that can guide changes in the group [Gersi1988a]. The midpoint transition is so important, even when not designated by the group as a time to alter behavior, that groups actually delayed implementation of changes until the midpoint even when the need for change was recognized and discussed earlier [Gersi1989a]. For large projects, other contextual cycles will supply additional times (such as a quarterly financial review) during which change is more easily accepted, or even expected. This influence of organizational rhythms, known as entrainment [Ancon1996a], provides natural intervention points:

“While entrainment suggests that the choice of which cycles to align is key to determining how successful a change intervention will be, it also suggests that ongoing paces and cycles will create windows of opportunity in which change will be more easily accomplished and other periods in which there will be more inertia and resistance.” [Ancon1996a]

It is also possible to schedule intervention points, by dividing the project into phases as Microsoft does [Cusum1995a, pg. 190]. By assessing fit in advance of the midpoint or completion of a phase, an intervention (if needed) can be planned to coincide with these natural change points. If project phases are “artificially” created to inject change opportunities, the project manager must be careful that workers accept phase endpoints as genuine (the character of the work changes) rather than seeing them as externally imposed interruptions in the work flow.

**Risk Assessment**

Risk assessment is the process of identifying risk factors, their likelihood of occurrence, and the likely severity of their effects. The identification process ordinarily is qualitative, consisting of a series of questions about potential risk sources. Often risk assessment is performed once at the beginning of a project, and then forgotten until the project post-mortem analysis. Risks that are focused on by the process in use are actively monitored and managed, while the degree of attention afforded to others depends upon the project management staff. If the project has major risks that are not monitored and managed by the normal process, the project is vulnerable. Only thoughtful project management and blind luck are working in the project’s favor for risk factors outside of the normal process’s scope.

There are a number of frameworks that can be used for risk assessment [Boehm1989a section 2, Shtub1994a]. Before using them, however, it is important to remember the goal: to focus the efforts of project management on those areas where it will do the most good. Unless working on safety-critical systems, listing every possible risk on a project will do little more than convince those involved of the futility of risk assessment. It is better to capture the top “several” (perhaps five) risks across the whole project and allow project management to focus on them, with periodic reassessments as well as current status. This can be done by surveying a cross-functional subset of the project
membership. It is important to have various functional areas represented in order to get the proper domain coverage.

The granularity of the assessment is also germane: it should be at a level high enough that similar risks are collapsed into a single category, but not so general that things with very dissimilar risk mitigation strategies are lumped together. Lists of risk factors identified on previous projects, often used as a checklist during risk assessment, can easily be misused. When used as survey input ("check box if this is a risk for the project"), such lists will bound respondents' thinking. It is preferable to simply collect everyone’s top five, then use the checklist to combine the responses into categories at the appropriate level and detect any omissions. One should also look for patterns in the responses: often, cultural factors are incorrectly assigned to other categories.

Following identification of the top few risks, severities can be assigned. In this context, it is the severity of negative consequences that are of interest. A high/medium/low scale is suggested, to avoid arguments over numerical values and to reinforce the notion that this is a qualitative analysis. The same scale can be used to assign relative probabilities of occurrence to each risk factor. Any serious debate over these high/medium/low values, or many medium values, signals a potential error in decomposition. If different parts of the project have different risk profiles, this is how it will be manifested. These different parts should be independently analyzed, as the different risk profiles might suggest different risk mitigation strategies that a "medium" rating for the overall project would disguise.

![Table of Risk Prioritization Example](image)

**Figure 3: Risk prioritization example**

Once the risk topography has been assessed, it is worthwhile to examine the whole for vulnerabilities to synergistic risks. Until candidate project architectures are proposed, the feedback loops which drive such risks cannot be fully identified, but often "likely suspects" – pieces of loops – can be identified early.
**Decomposition**

There is no simple rule for decomposition, yet it is one of the any architect’s most crucial decisions, precisely because it will influence the entire project. A “perfect” decomposition is often sought after, but rarely possible in practice. In the software field, formal education concentrates on functional decomposition, so applying functional decomposition to the product is a reasonable first choice. Decomposition according to criteria other than function allocation is, of course, possible. Remembering that the goal of partitioning here is to allow the parts to be produced using different processes and organizations if necessary, the most effective decomposition will be one that:

1. maximizes the partitioning of risk amongst inraproject categories (again, two categories with a value of low being best)

2. minimizes expected interface changes between the pieces (two stable interfaces are better than one interface expected to change considerably)

A partitioning that works for the project as a whole may nonetheless create functions that span the elements. These should be sought out when evaluating the efficacy of the project decomposition, in order to understand the cost of each candidate decomposition. If the candidate decomposition is adopted, element-spanning functions either should receive special project management attention or should trigger further adaptations of the project’s elements to eliminate them. If left unmanaged, the poor localized cleavage should be expected to result in downstream problems.

While easy to overlook and often ignored, organizational theorists have noted the existence of feedback loops involving structure. The organization within which the project is situated can constrain allowable decompositions: “...results obtained here emphasize that the structure and process choices for a particular organizational level are constrained and limited by design criteria imposed from macro-organizational levels” [Drazi1985a]. [Mille1992a] notes the existence of “accounts of mutual reinforcement among structural elements” leading to a “circle of influence” between these elements. He also notes “Structure can influence the intended rationality of decision-making processes, and vice versa...Intended rationality in decision making may well be a function of organizational structure” and “In a reversal of causality, process may influence structure” [Mille1992a].

**Adaptation and Fit Assessment**

Once a decomposition has been chosen, the various sub-projects can be evaluated to determine the element (process, product, organization) model which best matches the project requirements. Thus in a decomposition based on product functional allocation, process and organization models must be selected. The “best match” evaluation should be done without considering the company’s current competencies, and each element should be chosen independently to match the needs of each sub-project. Once target element models have been selected, the distance between company competencies and the targets are known. Actions to close the gaps can then be selected and applied, ideally removing the gaps entirely although this is not to be expected in practice. Standard risk
mitigation strategies, whose input includes the cost of each action, should be used. An example risk mitigation framework is shown below.

![Risk Mitigation Framework](image)

**Figure 4: Risk mitigation approach as a function of severity of consequences, probability of occurrence, and mitigation cost**

The congruence models presented in [Alexa1991a, Perez1994a, Perez1996a] can be used as starting points for fit assessment. As developed in the papers, they are designed for process model selection, but can easily be adapted for selection of other model types. The criteria used for evaluating fit in these models must be replaced with criteria relevant to the project’s context. The methodology of [Alexa1991a] combined with the notions of imperfect fit and adaptation from [Perez1994a, Perez1996a] and the possible addition of weights to each criterion yields a generalized selection process. This selection process is then customized to the task at hand by using the contextually relevant criteria described above.
Contextual Background

**System, Middleware, and Application Products**

The IBM division from which the cases are drawn deals exclusively with system software, in fact it deals only with the operating system software that has run mainframe computer systems since the 1960s. As is typical with operating systems, the degree of coupling between modules is higher than in application software, increasing the number of both desired and undesired interactions. Operating system software also executes in a wide variety of run-time environments while having higher reliability requirements than application software, making adequate testing that much more difficult.

Reliability and availability requirements are more stringent in system than application software, and this is particularly true of mainframes, which mediate access to large pools of resources for many users. Reliability is important since a single OS/390 system could easily be simultaneously serving a thousand time-sharing users, thousands of remote transactional database terminals, batch jobs, and HTML requests. The business impact of an unexpected outage is simply much larger than an outage of a smaller server. Further, when problems do occur, a reliable system restricts or prevents propagation of errors to unrelated parts of the system. Thus while a looping program might exhaust its available memory, other programs and the operating system itself should not be affected. Availability on the other hand deals with the issue of planned outages; while in-progress work will not be lost during a planned outage, the system is still not available for useful work. If outages of either type occur while computing demands are present, backup capacity requirements increase. Reliability and availability are two of the “ilities”: qualities such as reliability, availability, serviceability, scalability, and security, which provide market differentiation for this operating system. The expectation for new operating system and subsystem (database, transaction manager, et cetera) components is that they will, at worst, not diminish the existing level of the platform’s “ilities”.

Unlike most other server platforms, S/390 does not restrict a high-performance machine to running a single type of work (workload). Indeed, one strength of the platform is its ability to run multiple workloads simultaneously with superior performance. Each workload has distinct resource usage characteristics. A system that allows different work to compete for resources must have an effective set of mechanisms for mediating access to those (perhaps scarce) resources so that the most important work meets its performance requirements. Complex applications which support internal multiprocessing address similar issues, but the normal single-threaded application can ignore many of these problems. Those problems can be ignored by most applications precisely because the underlying layers provide the necessary abstractions.

Originally, only the underlying operating system that provided abstractions to applications. Over time, other layers developed which provided abstractions at a higher level. First, subsystems (more properly called resource managers) such as database managers and transaction managers were added to provide data coherency and atomicity
of updates. Later, middleware evolved whose main purpose was to act as a go-between, translating data formats between different applications and coordinating transactions which span resource managers. Middleware removed the necessity to have all data in the same resource manager while preserving transaction atomicity, allowing businesses to evolve their data processing methods without moving historical data to new formats as each new system was developed.

A distinction must be drawn between mainframes and the issues faced by software segments more familiar to the public, such as personal computers (PCs). Whereas most PC software receives bug fixes through upgrade or an infrequent “fix pack” containing the accumulated fixes for a variety of defects, mainframe system software produces an individual fix for each problem, and creates versions of each fix for all supported releases. Moreover, whereas PC software companies typically only support the latest release, mainframes typically support a release for five to ten years after first delivery to customers. There is a tremendous cost incentive therefore to deliver high-quality software.

Another important difference is in the area of architectural control and its effect on organizational competencies. System vendors, especially vertically integrated companies such as IBM in the mainframe arena, exert significant control on product architecture. This influences not only IBM's products, but also those of independent companies who use IBM's software platform as a base. As IBM's mainframe operating system evolves, so must those complementary products. Application vendors typically are free to build whatever users will buy, subject to the limitations of the underlying abstractions it uses. Middleware, which provides the interconnections between these two layers, must change to meet the evolving demands of both. Middleware architecture is heavily influenced by standards and dominant industry platforms, and so is not subject to the same level of owner control. Competition in applications and middleware is primarily function-based, whereas system-level products compete on both function and performance. The emphasis on performance favors task specialization, leading to deep competence in particular areas. Genuinely new competencies are less rarely required in system code than in application or middleware products.

**Product History**

In 1966 IBM released the System/360 (S/360) [Ausla1981a], in a “bet the company” gamble to provide general-purpose computer systems for business customers. Almost thirty-five years later:

- The hardware architecture has evolved from S/360 through S/370 and S/370 XA to S/390.
- The underlying hardware technology has transitioned from bipolar to CMOS.
- The operating system has evolved from OS/360 through MVT and MVS to OS/390.

Through all of these changes, one of the original tenets of S/360 – evolutionary change through object code backward compatibility – has protected the information technology investments of IBM's customers. IBM also garnered a significant number of technology
patents because of S/360 that gave it a near-monopoly in the mainframe segment throughout the 1970s. Even during the 1980s when plug-compatible manufacturers exerted pressure on mainframe margins, IBM’s mainframe market share remained quite high. Thus, the organization under scrutiny here evolved in an environment subject to less competition than most other parts of the computer industry. New functions were introduced according to a technology push strategy, and competition was based upon the “ities” rather than cost or features. Product release cycle times were typically three to five years, and major new functions were introduced in a “big bang” style rather than incrementally. The first release of a major new technology such as clustering (sysplex) contained all of the major functions necessary to build and support clusters; subsequent enhancements were primarily in the form of clustering exploitation.

The following diagram is a high level representation of the software normally running on an S/390 mainframe. It introduces a number of terms referenced here and in the cases, showing how the various pieces fit into the overall system.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Example Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>ISPF, Rexx</td>
</tr>
<tr>
<td>Application</td>
<td>SAP, QMF, CallPath</td>
</tr>
<tr>
<td>Middleware</td>
<td>MQSeries, CB/390</td>
</tr>
<tr>
<td>Subsystem</td>
<td>IMS/DB, DB2, Oracle</td>
</tr>
<tr>
<td>Transaction monitor</td>
<td>CICS, IMS/TM</td>
</tr>
<tr>
<td>System</td>
<td>OS/390, MVS, JES2</td>
</tr>
<tr>
<td>Operating system stack</td>
<td>DFP</td>
</tr>
</tbody>
</table>

**Figure 5: Simplified Mainframe Software Environment**

The S/390 software organization is responsible for products at the operating system level, as well as certain tightly integrated major subsystems. For historical reasons, the tightly integrated subsystems are not considered part of the actual operating system, although customers rarely see the distinction. Until the mid-1990s, customers ordered each product separately, although many were required to have an operational system. This was changed when more than forty products (including MVS) were repackaged into one orderable product (OS/390). Thus customers see a single OS/390 product, but product distinctions (MVS, JES2, etc.) are maintained within the organization. All of the case studies were drawn from projects at the site in Poughkeepsie, New York.
Figure 6: OS/390 and MVS Structural Overview

The first three cases are drawn from projects in the performance management area of MVS and OS/390, so this area is shown in greater detail. System performance is managed by a software component named the System Resource Manager (SRM), which was added to MVS during the mid-1970’s. The overall design of this component is to accept a set of instructions from the system programmer, which details three prongs of the installation’s performance policy:

1. How to divide work into groups.
2. How to allocate each resource controlled by SRM to each group of work.
3. System-wide settings not specific to any one group of work.

SRM’s goals are to implement the installation’s performance policy, and within the bounds of that policy to maximize system throughput. SRM accomplishes this by controlling access to the computer’s central processing unit (CPU), to memory, and to disk devices. Its overall results, however, are an emergent property of the underlying resource management decisions rather than the simple composition of them. This emergent behavior is an important point that will resurface later.

Process History

[Brook1975a] and [Radic1985a] describe IBM’s software development processes during the 1960s and 1970s; the overall processes have been remarkably stable through the 1990s. To be sure, implementation languages have changed and particular tools have come and gone, but the basic waterfall process is still intact. This reflects in large part the business environment, which has not exerted the degree of pressures for fundamental change seen in the microcomputer and Internet segments. As in the 1970s, the predominant metrics used for schedule estimation are size (lines of code) and effort (programmer-months). The waterfall process is used, along with a number of formal coordination mechanisms, to reduce process variability as much as possible. Most process measures in use focus on performance (primarily schedule) of a product against a plan. Phase completion milestones, such as DCUT (design, code, unit test complete), are the predominant measures of progress; interim project monitoring checkpoints use best-guess progress estimates (planned vs. actual, where planned is a moving target and actual is typically difficult to measure). Release dates within which each project must work are determined during the planning process; prior to the early 1990s, these dates were
malleable. With the transition from MVS to OS/390 in the early 1990s, however, OS/390 release dates are now prescribed and completely unmovable.

Risk management is usually handled by reallocating personnel, and to a lesser degree through product re-scoping. The waterfall process is thoroughly ingrained in the culture; its use is rarely questioned, and even less rarely are other process options officially used. There is a strong aversion by management to de-commit to delivery of any item once it has been booked in the product plan. A complex web of inter-product dependencies exists between development sites, and any decision to de-commit not only opens the messenger to criticism but also allows other projects dependent on the decommitted item to alter their plans. Thus timely delivery, especially with respect to commitments to other sites, is a high priority.

Figure 7: Modified waterfall process overview.

In the figure above, time and projects flow from left to right, although the process stages are not drawn to scale. The difference between a true waterfall process and a modified waterfall is the degree of overlap between stages, with the modified waterfall having more overlap. The tables below, based on [Radic1985a] and the author’s experience, provide finer process resolution and relevant details about each stage.
Table 1: Mapping of waterfall process stages to IBM programming process architecture

<table>
<thead>
<tr>
<th>Family</th>
<th>Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements &amp; Planning</td>
<td>Requirements &amp; Planning</td>
</tr>
<tr>
<td>Design</td>
<td>Product Level Design</td>
</tr>
<tr>
<td></td>
<td>Component Level Design</td>
</tr>
<tr>
<td></td>
<td>Module Level Design</td>
</tr>
<tr>
<td>Implementation</td>
<td>Code</td>
</tr>
<tr>
<td></td>
<td>Unit Test</td>
</tr>
<tr>
<td>Testing</td>
<td>Functional Verification Test</td>
</tr>
<tr>
<td></td>
<td>Product Verification Test</td>
</tr>
<tr>
<td></td>
<td>System Verification Test</td>
</tr>
<tr>
<td>Packaging &amp; Validation</td>
<td>Package and Release</td>
</tr>
<tr>
<td></td>
<td>Early Support Program</td>
</tr>
<tr>
<td>General Availability</td>
<td>General Availability</td>
</tr>
</tbody>
</table>

Table 2: Process Step Definitions

<table>
<thead>
<tr>
<th>Process Step</th>
<th>Entails</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements &amp; Planning</td>
<td>Product and system level requirements are documented, and project planning is begun.</td>
</tr>
<tr>
<td>Product level design</td>
<td>Allocation/decomposition of functions to layers of the system and to system components (either new or existing), translation of business requirements into functional requirements. Definition of inter-product and important inter-component interfaces. S/390 calls this System Level Design.</td>
</tr>
<tr>
<td>Component level design</td>
<td>Decomposition of a single component’s function into sub-components, allocation of functions to sub-components. Detailed design of inter-product and inter-component interfaces. S/390 calls this High Level Design.</td>
</tr>
<tr>
<td>Module level design</td>
<td>Decomposition of sub-components into individual modules, allocation of function to modules. S/390 calls this Low Level Design.</td>
</tr>
<tr>
<td>Code</td>
<td>Implementation of low level design.</td>
</tr>
<tr>
<td>Unit test</td>
<td>Pre-integration testing of each module, typically using a simulator.</td>
</tr>
<tr>
<td>Build</td>
<td>Integration of unit tested modules into the product.</td>
</tr>
<tr>
<td>Functional Verification test</td>
<td>Functional white/grey-box testing, concentrating on interfaces internal and external to the component, typically using a simulator. S/390 calls this Component</td>
</tr>
<tr>
<td>Test Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>System Verification test</td>
<td>Functional black box testing, including regression testing and stress testing using real hardware. S/390 calls this System test.</td>
</tr>
<tr>
<td>Product Verification test</td>
<td>Pre-defined environments used to generate performance data to evaluate the release-to-release overhead changes and feed that information back into development for correction of any major increases. Concentration is on a specific small set of unconstrained environments; exploitation of new function is excluded. If new performance-oriented functions are in the release, special performance studies are run to collect data for announcement. S/390 calls this Performance Test.</td>
</tr>
<tr>
<td>Package &amp; Release</td>
<td>Merging of multiple products into a “stack”, or group of tapes and publications that together can be installed as a system.</td>
</tr>
<tr>
<td>Early Support Program</td>
<td>Availability of the product to a limited set of early (beta test) customers.</td>
</tr>
<tr>
<td>General Availability</td>
<td>Diagnosis of problems reported by customers.</td>
</tr>
</tbody>
</table>

A cultural preference within IBM for all-encompassing solutions and grand architectural visions rather than incremental function delivery supported cycle times averaging three to five years. As the cycle time increased, however, the tendency for customer needs and requirements to shift during the development cycle increased, causing rework and in some cases missed market windows. To reduce these effects, limited parallelism was introduced into the waterfall process.

In a time-scaled representation, one would see that a significant portion of the elapsed time required to create a new release is spent in testing. 50% is a common rule of thumb for new releases, with a majority of the elapsed time (as distinguished from person-time) being spent in various levels of integration testing. This is common as software products evolve over time, due to concerns about compatibility. The mainframe software arena has a particularly virulent case of this problem due to its long history, and due to the fact that crisp definition of the interfaces intended for use by software outside of the operating system was not provided until the late 1980s. Consequently, many software packages provided by IBM and other companies made detailed assumptions about the inner workings of the operating system. These assumptions represent implicit coupling of the operating system with outside software, so the whole is more “brittle” or liable to unintentional breakage by changes than would otherwise be the case.

While many software process models ignore the maintenance process, within IBM diagnosis and post-manufacturing defect removal receive special attention. During the 1980s, the service process was isolated from product development, both from a process view and organizationally. Fixes to field defects were created by the same personnel who performed diagnosis, while programmers working on succeeding releases merged this
stream of fixes into the code base. This led to a predictable set of communication and coordination problems between the two groups; in the early 1990s, the responsibility for creating field defect fixes was moved back into the groups doing development.

**Release Content Planning**

Personnel resources are allocated based on a semiannual plan cycle that occurs in the spring and fall. The fall plan lays out planned expenditures over the next year, while the spring plan has the character of a mid-course adjustment. A team including the system level designers and other upstream groups prioritizes items, and bottom-up resource assessment follows to map resource requirements to items. After a negotiation period, the top projects are funded and the rest deferred to the next plan cycle.

Once the release plan is built, items are mapped to projects that move downstream from design to development, then to system test and so on throughout the rest of the process. As shown in the next section, organizational boundaries normally delimit projects. Each functional area (discipline) creates its own project plan for each project, to execute the phase(s) of the process for which it is responsible. Thus the development project plan includes only the low level design through component test process phases; system test and performance test create their own plans for testing each project, which often have some coverage overlap. The dozens of projects required for a typical release proceed concurrently through the process, but in the aggregate most projects will be at the same process stage at the same time. The staged delivery to customers and the relatively serial process interact to create a “bulge” of effort that moves downstream through the system once for each release. Personnel resources utilization is leveled by having a sequence of releases in progress; as soon as the design group ramps down its involvement with release “n”, it begins on release “n+1” and so on. Written project post-mortem analyses are not used; knowledge transfer between projects is by interpersonal contact, usually due to staff movement over time.

**Organization History**

The existing organizational structure was platform-based at the highest levels, containing strong product and functional divisions within each platform. In 1998, the S/390 software organization at the Poughkeepsie site had approximately 900 full time employees distributed between design (40), development (400), build (35), system test/performance test (300), and service (150). These people were responsible for several OS/390 elements: MVS, JES2, JES3, RACF, LE/370, SDSF, CB/390, and others.

[Clark1992a] provides a taxonomy of team types; those listed in the following table will be used here. [Ulrich1995a] provides a similar summary, with further comparative elements. The descriptions below are excerpts from [Clark1992a].

| Functional | People are grouped principally by discipline, each working under the direction of a specialized subfunction manager and a senior functional manager. The different subfunctions and functions coordinate ideas through detailed specifications all parties agree to at the outset, ... Over time, primary responsibility for the project passes sequentially...from |
one function to the next...those managers who control the project's resources also control task performance in their functional area; thus, responsibility and authority are usually aligned. ... The associated disadvantage is that individual contributions to a development project tend to be judged largely independently of overall project success.

Lightweight Like the functional structure, those assigned to the lightweight team reside physically in their functional areas, but each functional organization designates a liaison person to "represent" it on a project coordinating committee. These liaison representatives work with a "lightweight project manager," ...who coordinates different functions' activities. ... The project manager is lightweight in two important respects. First, he or she is generally a middle- or junior-level person,... usually has little status or influence in the organization. Second, ... the key resources (including engineers on the project) remain under the control of their respective functional managers. The lightweight project manager does not have the power to reassign people or reallocate resources.

Heavyweight ...the heavyweight project manager has direct access to and responsibility for the work of all those involved in the project. Such leaders are "heavyweights" in two respects. First, they are senior managers within the organization; they may even outrank the functional managers. ... Second, heavyweight leaders have primary influence over the people working on the development effort...However, the longer-term career development of the individual contributors continues to rest not with the project leader...but with the functional manager...

Functional divisions dominate all aspects of S/390, to the point that it causes confusion to those outside this organization. When used in a product context, functional areas (or components) correspond to the traditional behavioral decomposition familiar to those trained in software. Using this style of decomposition, an electronic mail system might be divided into text editing, name resolution, organization (file folders or categories), and communication functions/components. When used in a process context, functions refer to process steps like coding and low-level design. When used in an organizational context, functional areas refer to the organizational unit(s) responsible for a particular process step (a discipline). The S/390 software organization is subdivided by platform, OS/390 being one of them. The OS/390 software organization is subdivided into design, development, build, system test, and performance test, each of which is responsible for one or more process steps. OS/390 development is further subdivided along product lines, so that a single first line department manages a collection of one or more components. Thus, the functional organization is simply the formal organization as described by a standard organization reporting chart.
<table>
<thead>
<tr>
<th>Functional Area</th>
<th>Process Steps Performed, Personnel Allocation Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>System level design, and component (high) level design in conjunction with component developers. One design group at this level per product, e.g. MVS. Strong functional division from most other groups, moderately strong division from development.</td>
</tr>
<tr>
<td>Development</td>
<td>Low level design, code, unit test of new code and fixes. Personnel typically specialize in one component for several years at a time, each most often working on a single release at time. Senior members usually gravitate toward design, where more release parallelism is required. Strength of functional division with component test dependent largely upon current organizational structure, moderate with design and build, and strong with other areas. Depending upon the period, component test might either be performed by the same or different departments than development. The organization oscillated between segregating component test and development into independent departments and merging them. The degree of independence between development and component test is a long-running organizational tension.</td>
</tr>
<tr>
<td>Build</td>
<td>Build. Integration of code into builds which are used for downstream processes: component test, system test, performance test, product manufacturing. Personnel are allocated by product release, and may work on several different products and/or releases in parallel. Moderate functional division with other areas.</td>
</tr>
<tr>
<td>Component test</td>
<td>Component test. Similar to development in personnel allocation behavior, with senior members creating test plans and acting as test team leaders. Strength of functional division with development dependent largely upon current organizational structure (see note above under Development for details). Moderate functional division with development and build, strong with other areas.</td>
</tr>
</tbody>
</table>
| System test     | System test. One group per product, with most personnel assigned to one release at a time. For large products such as MVS, personnel are assigned to concentrate on particular sections (components) of the product. Problem diagnosis is done by a separate debug group that works across products and releases. Strong functional division with areas other than Build. During the 1990’s, some of the historical separation between system test and performance test was eroded by a
The formal organization was structured around disciplines known as functional areas (left column, preceding table), with product subdivisions inside this organization. Thus, "development" as a function meant that all personnel primarily concerned with low level design, code, and component test of a particular operating system component belonged to the same organizational unit. Within the development organization, departments were dedicated to individual products (RACF, CB/390, OS/390) or OS/390 components (Workload Manager, Unix System Services). Department managers at the lowest level were referred to as "first line" managers, and so on as shown in the following figure. Other software platforms in the S/390 family report to the VP of S/390 software development, but are otherwise irrelevant to the case studies and omitted from the figure.

![S/390 Formal organization overview](image)

**Figure 8: S/390 Formal organization overview**
Once release content was selected, a former functional manager would be appointed as the release manager to coordinate and be responsible for delivery of the release. Release content was decomposed down to the component level and grouped into projects as shown in the preceding figure. Previously existing components were each owned by a single development department; a new component would either be added to an existing department’s responsibilities or have a new department created to own it. In the case of a large new component, as many as five new departments might be created under a single second line manager. Ownership of a component was assigned to the lowest development manager in the hierarchy whose span of control included all departments working on the component. When a project required changes to multiple components, it was divided into separate projects partitioned by component owner. Although an experienced manager (unlike [Clark 1992a]’s lightweight manager), the release manager was lightweight since authority to reassign personnel remained with the functional development managers. A site-wide project office assisted release managers in coordinating dependencies between components and with products both inside and outside the laboratory. Management of individual projects consisted of the component owner (a first or second line manager) and one of the most experienced component technical experts reporting to that manager.

**Incentives**

The performance evaluation system in use individually rates employees to determine contribution, which is then translated into pay. Appraisal periods are one year in duration, with partial-year periods when major delimiting events such as promotions or a change in manager occur. Goals are jointly set by the employee’s functional manager and the employee; ratings are done by the functional manager, with higher level reviews to assure consistency across the larger organization. At appraisal time, the rating is
discussed with the employee, and comments solicited. Since the down-sizing in 1993, certain aspects of the system have been changed; the employee takes a larger role in setting performance goals, a set of team goals is now expected, and peer reviews play a larger role in the review process. The underlying influence of functional management in this process however is unchanged.

**Conflict Resolution**

Conflict resolution has both formal and informal components, with the large majority being solved via informal mechanisms. In the simplest case, the two parties manage to strike an agreement without outside help. If agreement cannot be reached, or in decisions affecting resource (personnel) allocation, the employees' functional managers become involved. Prior to the down-sizing in 1993, the resulting management resolution process – which often involved multiple levels of management in order to reach the point where a single individual had authority over both parties – was quite lengthy, and called “escalation”. As the organization has been flattened, escalation has largely fallen out of favor. The point of control is typically closer to the bottom of the organizational hierarchy, so fewer levels of management get involved and fewer managers are inclined to waste time on turf battles.

A persistent source of tension in this organization is that between aggressive delivery of new function and quality concerns. Upper management and designers, who are part of the selection process which determines release content, tend to emphasize function delivery. Since developers work closely with the designers as the project proceeds into development, they tend to share this attitude. Middle and first line development management, who are measured on execution of the plan (on-time delivery) and product quality (as measured by defects), tend to emphasize adherence to the waterfall process which by cultural association translates into the “ities”. Projects where functional delivery is more (or as) important as the “ities” are rarely selected by this organization, but when they are the differences between the two groups’ behavior becomes evident.
Case Study Overview

The case studies presented span a range of projects within S/390 software; a comparison table showing major project attributes across the studies is shown below. An important distinguishing feature of CB/390 is its status as a full product, whereas the others are incremental enhancements to an existing product.

Table 4: Case Study Comparison

<table>
<thead>
<tr>
<th>Project</th>
<th>WSM</th>
<th>WLM1</th>
<th>WLM4</th>
<th>CB/390</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (scaled lines of code)</td>
<td>1</td>
<td>13</td>
<td>3</td>
<td>83</td>
</tr>
<tr>
<td>Effort (scaled programmer-months)</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>Duration (elapsed time in months)</td>
<td>14</td>
<td>33</td>
<td>15</td>
<td>32</td>
</tr>
<tr>
<td>Number of IBM products with interproduct interfaces</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Number of non-IBM products with known interproduct interfaces</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Product</td>
<td>MVS</td>
<td>MVS</td>
<td>OS/390</td>
<td>CB/390</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of distinct development processes used</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project Organization</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(F=Functional, L=Lightweight, H=Heavyweight)</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F &amp; L-PDT</td>
</tr>
<tr>
<td>Final formal organizational structure</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>H &amp; L-PDT</td>
</tr>
<tr>
<td>Initial informal organizational structure</td>
<td>F</td>
<td>H</td>
<td>H</td>
<td>F</td>
</tr>
<tr>
<td>Final informal organizational structure</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>Number of dedicated project departments</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Number of departments providing resources</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Number of IBM geographic locations with product interfaces</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Major Anticipated Risk Factors (estimated by thesis author)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
</tr>
<tr>
<td>Requirements</td>
</tr>
<tr>
<td>Innovation</td>
</tr>
<tr>
<td>Schedule</td>
</tr>
<tr>
<td>Process</td>
</tr>
<tr>
<td>Organization</td>
</tr>
<tr>
<td>Environmental</td>
</tr>
</tbody>
</table>

A common position taken by project managers is that their project is “unique”, that lessons from earlier projects are either too specific to be useful or too general to be relevant. While there is a kernel of truth in this, its application can easily be too broad. While everyone would like to believe that their work is totally, genuinely new, the truth is there are sources of learning accessible to those willing to search. Conclusions drawn here, because of the case studies selected, apply most easily to the organization from which they were selected. Their more general application requires an understanding of the assumptions underlying the business and their effects on project selection, process, and organization. Key to eliciting a usable risk analysis for comparative purposes is the proper level of abstraction. Too detailed of an analysis leads to results that transfer poorly across projects, while too aggregate of an analysis delivers little useful information.
**Major Risk Categories**

A recurring theme throughout the cases is the interaction of product, process, and organization, which can be likened to the three legs of a stool. Of course, the environment also plays a role: expected actions of competitors, evolution of standards, organizational commitment to the project, and the like. To these, another is added: synergistic risks, which arise from the interaction of the other structures. Synergistic risks are consequences, often unexpected, that emerge from the system as a whole rather than from any single aspect taken alone. To extend the stool analogy, they represent the arrangement of legs on the stool. Place all three legs in the center, and most users will not consider the stool a good one, although all of the pieces are present in the correct proportions. The notion of structure causing behavior is taken from the field of System Dynamics: “Until recently we have been insufficiently aware of the effect of time delays, amplification, and structure on the dynamic behavior of a system. We are coming to realize that the interactions between system components can be more important than the components themselves” [Forre1961a]. The reader is referred to texts in that field for further discussion of its details and concepts.

Product risk consists of areas such as the degree of innovation required in the solution, the rate of requirements change, and schedule pressure (which is actually a combination of estimated size, staffing, and normal process throughput). These are the factors that the organization under study is accustomed to managing through its normal development process. Projects are normally incremental changes to existing components of existing products, and the implementation is reasonably well understood at a high level before the project is staffed. The case studies presented were specifically chosen because they were recognized to have unusually high levels of product risk relative to the average project undertaken by the S/390 software organization.

Process risk usually arises from the introduction of new process technology: computer-aided software engineering (CASE) tools, new source code languages, and so on. Less frequently, basic changes to the development process itself are made. Historically in this organization relatively little emphasis has been placed on development productivity tools, so that projects breaking new process ground often must cope with a lack of tool support.

Organization risk describes the degree of fit between the project and organization structure. This organization has a 30-year history of strong functional alignment, and corresponding “stove pipe” culture, which has eroded somewhat since the 1993 IBM layoffs. Included here are risks associated with culture, such as an organizational culture aligned to quality that has trouble shifting orientation on a project where schedule is more important than initial quality. The notion of organizational distance – the degree of communication difficulty experienced when crossing various organizational boundaries – also falls into this category.

Environmental risk encompasses aspects such as the degree of organizational commitment to the product family or platform, alignment between the project and product strategy, the real or anticipated actions of competitors, and the degree of architectural control that the project exercises. While none of these is directly related to
the project architecture, they nonetheless influence and in some cases constrain behaviors that affect the project as a system.

**Data Collection Methods**

The primary source of data was the author's personal experience, so there is built-in collection bias. The author worked on the first three case study projects: as a component test lead in WSM, developer and overall test lead on WLM1, and as overall project leader for WLM4. Personal recollections, electronic mail, archived forums, historical project plans, and interviews with other members of each project rounded out the data sources on these cases. The author never personally worked on the CB/390 project. Data on this project was collected via extensive interviews with one of the original architects, a risk assessment survey given to project team members who helped shape the project from its inception, an hour-long interview with the then-current heavyweight project manager, and several smaller informal interviews and electronic correspondence with past members of design (architects) and development. Included in the interviews was one person who had moved to CB/390 from DSOM. Since CB/390 is not complete as of this writing, project totals of size, effort, and duration are estimates based upon the current project plan.
Case Study: Working Set Management

Working Set Management (WSM) consisted of a set of new functions (features) added to an existing component (SRM) of an existing product (MVS). It would generate no direct incremental revenue, although it was expected to drive additional demand in the numerically intensive computing market.

Background

Working Set Management was the name given to a new class of storage management algorithms, developed in response to requirements from the scientific computing community. Numerically intensive work degraded overall system performance when run concurrently with traditional workloads: interactive users, batch jobs, and on-line transaction processing. This degradation was due to specific patterns of memory access that caused significant paging overhead in MVS. For a more in-depth discussion of WSM, see [SlyPg1991a, Eiler1990a].

Stakeholders

The ultimate stakeholders for this project were the high-end customers involved with scientific and numerically intensive computing applications, such as Boeing. Performance estimates indicated that these customers could see an order of magnitude improvement in the elapsed time required to run jobs performing tasks such as analysis of airflow over a wing. IBM's Numerically Intensive Computing (NIC) group and Marketing had significant incentives, as they projected the potential to capture several hundred million dollars worth of incremental revenue by improving MVS's ability to manage these jobs.

MVS product management in Poughkeepsie, NY also had a significant stake in WSM. Management was a conservative group, yet WSM was added to the release plan very late in the product cycle due to its potential revenue impact. The MVS release date would be announced before all of the code was even written; moving a release date after making a public commitment was considered an unacceptable alternative. Compounding things, this particular release of MVS was large, containing enough new functionality to justify a price increase. Since the MVS product set generated a substantial portion of IBM's cash flow, a delay in the release would meet with heavy scrutiny from executives.

Each of the functional area managers -- design, development, build, system test, and performance test -- was under pressure from upper management to add WSM very late in the product cycle. It was expected to require more system test and performance test resources than traditional projects, and was recognized to have significant risks. The MVS release had already begun system test when WSM was added to the release plan, and system test historically struggled with resource over-commitment as they had to absorb all upstream schedule slips in the face of an entrenched release date.
Primary Risks Addressed by Project Plan

The financial risk of foregone revenues operated at the highest management levels, resulting in the addition of the project very late in the release cycle. The downside risk of delaying the entire release was addressed by isolating the code in a separate branch of the library management system, allowing it to either be merged into the target release (if ready), or shipped in the next release. This required extra work to write, build, and test the code, but the relative cost was minimal compared to the risk of degrading the quality of the release base through premature integration. There was also an intermediate option of disabling the code before delivery to manufacturing and later enabling it as part of release maintenance after final fixes were tested, which was not pursued. While technically possible, this option was deemed too disruptive to downstream schedules.

Schedule risk was the overriding concern once the project had been added to the release plan, but this was actually more of an emergent property of the project than a concern about its size. The controllable portions of this risk were addressed by the staged delivery that allowed overlap of the code/unit test and component test phases, using the normal modified waterfall process.

There was significant innovation/technology risk in the product content: while the new storage management algorithms worked in theory, they had never been validated in practice. In addition, an IBM-internal object-oriented computer-aided software engineering (CASE) tool was being used to implement those portions of the code that required entirely new modules, adding process risk. While this tool had undergone testing, including by one of the WSM designers, it had never been used for an actual MVS project before. On the surface, this appears to be an unwarranted risk, but testing had shown both a significant increase in coding productivity and a reduction in error rates using this tool. The impact of CASE was thought to be confined to low level design and code since it generated the same source language that coders normally worked in, although feedback indicated downstream impacts in perceived serviceability.

Finally, there was substantial process risk when dealing with SRM modifications. History had shown that the normal development process allowed a significantly higher number of SRM defects to escape component test relative to other MVS components, so SRM problems in general tended to be found later in the process. Some of this was blamed on timing effects; component test was performed while running a test MVS image under a time-shared virtual machine (VM) simulator, while downstream test processes and customers all ran on real hardware which exhibited different timing and multiprocessing behavior. Debugging under VM was much easier, and its costs were significantly lower than hardware testing, but at the cost of increased defect escapes. A large portion of the problem, however, was the scope of each test phase: component test concentrates on intra-component functions and external component interfaces, system test on stress testing, and performance test on gathering data for performance analysis. SRM has relatively few external interfaces and operates mostly through controlling performance by directing MVS’s physical resource managers. Thus, component test can reliably verify relatively few SRM code paths, system test tends to detect SRM coding problems but not design problems, and only performance test considers the scope.
necessary to discover more subtle design defects. Even performance test is not perfectly matched to SRM’s test requirements, as it concentrates on measuring the behavior of a handful of specific environments for publication rather than realistic customer run-time environments.

**Product Plan**

The implementation plan for WSM was to use existing SRM infrastructure as much as possible, add code to detect when WSM management was necessary, and to implement the related storage management decisions in a new subcomponent of SRM. Concurrent changes were required in the MVS memory manager component, and the system performance monitoring product. All software products involved were owned by groups geographically co-located within 20 miles of Poughkeepsie, NY, USA. Compared to most MVS projects of the time, WSM was estimated to be small and had few changes to externals.

After the specification was completed, a three-stage delivery sequence was planned to allow overlap of the development and component test phases. Stage 1 would consist of basic infrastructure with SRM and two closely related MVS components containing minor corequisite changes. Stage 2 would consist of new SRM infrastructure to collect the data that would be used as a basis for decision-making in WSM’s algorithms. Stage 3 would consist of the algorithms themselves.

**Process Plan**

The plan was to follow the normal MVS development process, a modified waterfall model. As each of the three stages was built, test variations would be created and then run while low level design and coding of the next stage was underway. Experienced testers unfamiliar with SRM were staffed as overall component test leaders after the project specification was completed. Their job was to create a test plan, which was completed concurrently with code delivery of stage 1.

The test plan was unusual in several ways. Its scope was expanded beyond the normal component test boundary of intra-SRM interface and functional testing. Stress testing on real hardware, normally the purview of system test, and limited performance test components were added in order to gauge satisfaction of the goal of improving overall system performance in the presence of mixed numerically intensive workloads without degrading performance in traditional environments. The system test and performance test functional groups formulated independent test plans, but they were coordinated to minimize coverage overlap.

The test plan also included the creation of a new internal testing tool. While certain system-level performance measures were available in the system test and performance test environments, this data had two problems: not enough information was available to perform problem diagnosis, and the metrics were not available in real time. In addition to supporting problem diagnosis, the tool could provide real-time presentation of SRM-specific information that would be useful in isolating regions of the multi-hour tests where particularly interesting emergent behavior was exhibited. This could then guide
examination of the detailed internal data collected by the tool on a timed interval. As a frame of reference for the reader, test sessions on hardware could run up to eight hours in duration and WSM was making storage management decisions every two seconds.

![Diagram of MVS release, system test, performance test, WSM project, stages 1-3, plan, low level design, code, unit test, component test, system test, algorithm test timeline from Nov to Feb.]

Figure 10: Working Set Manager Schedule Overview

**Organization Plan**

The WSM project was unusual from the start due to its intensive involvement by groups outside of development and its members’ willingness to cross-organizational boundaries. Throughout the development process there was involvement from the Numerically Intensive Computing (NIC) group, and from experts in design and performance analysis. NIC personnel provided examples of jobs that WSM was intended to manage better, which were tunable to provide different levels of storage usage so that they could be matched to test machine configurations. They also conducted their own testing of certain externals, and participated in test planning.

The design experts actually coded and helped test significant portions of the WSM algorithms; normally they would have provided only high level designs and then been called in when problems arose downstream in the process. The two test leaders, previously unfamiliar with SRM, were to supervise component testers more experienced with SRM. Both test leaders were familiar with the process of building code into systems; one was a veteran of the MVS release build group, the other had previously written code, been a component test team leader, and built systems for pre-integration testing of new code by coders. The organizational dispersion of the project members was large, as can be seen in the following figure. While this did not cause immediate problems even with the expanded test scope, eventually these stresses coupled with the process changes that evolved later became a major issue.
Figure 11: Working Set Manager Organization Overview

**Project Evolution**

The WSM project schedule was optimistic, although not unusually so for this organization. The designers and developers created a specification, and estimated the development (low level design through component test) effort required to deliver the project using assumptions standard for this organization: waterfall process, well-understood functional content, and standard LOC (code size)-based code/test productivity rates. The overall MVS development functional manager was aware of the innovation risk, and treated this project as an experimental step toward a different MVS function delivery paradigm. This awareness led to the creation of a contingency plan (move the function into the next MVS release) before the project started.

As the component test leaders increased their understanding of SRM and increased the proposed scope of testing, system test and performance test were brought into the team. A person from each functional area was dedicated to coordinate the test plans, and to provide access to hardware for early testing in each environment. Such an integrated test approach was very unusual in MVS development due to the strong functional orientation of the organization. The various functional test groups quickly evolved into a *de facto* project test team with high internal cohesion. The NIC group also acted as distributed members of this team through their efforts to develop performance benchmarks for use during product announcement. To facilitate communications, the test leads created a commonly accessible electronic project forum used for routine project communications between all parties, including those such as the build group whose orientation was still primarily functional. This replaced the heavy use of electronic mail, with its associated problems of changing distribution lists that caused some members to miss important information, and created a permanent record of project communications. As the coding of Stage 3 was completed, the designers became increasingly involved with observing the tests of the earlier code. Based on discussions with the testers, several design changes were made. For example, the need to understand the flow of WSM decisions leading to an emergent behavior in order to perform problem diagnosis led the designers to add an internal decision trace mechanism. Later as the project progressed, similar feedback loops occurred.
The execution of the project proceeded as described above until the third and final stage of code was built. The first practical tests of SRM's new algorithms showed that they did not work robustly at the system level as expected, although they passed the system stress tests. Management exercised the contingency plan to move the code into the following MVS release. While this was very disappointing to the project team, most found this a reasonable decision, since the required algorithmic solutions were unclear. This represented a six-month delay in customer availability, but internal milestones were such that delivery within the new dates was not assured. Since WSM's original schedule was later in its original MVS release than normal, the six-month customer availability delay translated into an additional three months of development time.

The designers, who had been working closely with the test leaders during tests on native hardware, had ideas for fixes. The test leaders had experience creating test code builds, and the ability to run tests that could feed information for further refinements back into the design. While the functional development group worked on moving the WSM code into the new release, the designers and those involved with hardware tests evolved into a new team concentrating on correcting the underlying algorithms. This team made significant progress in a short time by working on one subclass of problem at a time, in priority order. A design change would be discussed and finalized, coded by the designers, and built/tested by the testers. Testing proceeded from VM simulation to system test to performance test based on the information from each stage, as both the designers and the testers attended test shots. The testers provided their own process tools support, and official support channels were used for the CASE development tool. Using the traditional waterfall development process to do this would normally take one to two months per iteration; after an initial startup period, this informal group generated eighteen iterations in nine weeks. On several occasions, two iterations per day were completed from initial design through performance test. Iteration was driven by function and staging however, not time; none of the participants was concerned with the elapsed time values. All were concerned with testing the current iteration, what functions should be delivered next, and which functions could be delivered in time for the next hardware test shift.

**Figure 12: Iteration process flow**

While the process performance increase was substantial, this process was not universally accepted and never formally agreed to by development. After the first two iterations, serious conflicts arose between functional areas that resulted in a six-week project freeze while it was escalated several levels up the functional management chain for resolution. The essence of the conflict was between those who wanted to revert to the normal waterfall process for WSM, and those who saw the iterative approach as the only possibility for delivering the code with correct function on a reasonable schedule. The
functional incentives of the development organization heavily favored risk aversion, which within this organizational culture meant the traditional waterfall process. The designers, operating with a higher level view of getting the product right and then into the field as quickly as possible, disagreed. While the iteration team could continue to enhance the code’s functionality on its own, the (disjoint) development team could prevent final integration of the code into the product. Resolution of this required a decision from the laboratory director, changing the project management structure to an informal heavyweight project management structure. The iteration team gained ownership of the project, and the functional group was required to integrate the final iteration results into the product base. Had the implicit project and functional organization issues been addressed more thoroughly when the project started, this might have been avoided.

![Diagram](image)

**Figure 13: Working Set Manager schedule overview, showing iterations**

**Discussion**

The preceding addresses the surface facts of the project, but in attempting to draw conclusions for wider application some of the deeper issues and supporting information must also be examined.

The essential point of this case is the relationship between product, process, and organization. A significant departure from the process ordinarily used by this organization occurred, in response to the product risk profile, which set it in opposition to the functional organization structure. Iteration is a natural way to address technology and schedule risks by gathering more information earlier and validating central assumptions earlier in the process. In this case, the stage three build showed that the algorithms did in fact manage synthetic jobs designed to stress overall storage management reasonably well, but the algorithms failed to adequately manage actual customer jobs, which exhibit less consistent behavior. Thus the central idea of the patent was supported, but its model failed to account for several types of temporal variation within jobs. The early testing on native hardware, while quite expensive compared to VM simulation, accomplished a number of goals. First, it provided experience with the algorithms’ behavior on actual hardware, which was important since the designers expected to see significant timing differences between the simulation and real hardware. Second, it allowed testing the algorithms with jobs more realistic than the pure synthetics used to stress storage.
management traditionally. Third, it flushed out bugs much earlier in the process than historical data would predict.

The iteration process however also greatly increased organizational stresses, because the control of the functional organization was threatened. While most of the people either ignored organizational boundaries or developed processes (e.g. the forum) to make them irrelevant, the reporting and incentive structure was still oriented around and implemented by functional management. The point in the management hierarchy at which (most) team members were included was the lab director, who was responsible for 1,500 people. Once disagreements developed, the resulting escalation process wasted six weeks and seriously affected morale amongst and in some cases between the technical people involved. Most project communications were informal in nature, facilitated by the majority of people residing within the same building, but this reflected the existing office arrangements rather than a conscious plan to address communication issues.

**Lessons**

Iteration is a valid way to address technology risk by gathering information as early as possible; a critical measure, although soft in nature, is convergence. If the technical solution is improving, it is reasonable to continue funding as long as the business’s investment criteria are being met. If changes are undirected or if convergence on one aspect of the problem routinely degrades other aspects, the project may be out of control.

Problems are found earlier using iteration rather than a waterfall process, so the learning process is begun earlier. The time constants involved are ordinarily shorter than a waterfall process; all processes upon which continued iteration depends must be considered, since the slowest step will gate the process cycle time. Problem diagnosis and fix times are key areas.

Several enabling practices underpin any attempt to employ iteration. Lest one be tempted to decide to iterate without considering the implication on other areas, listed below are activities which were central to implementing iteration on this project and which should be addressed in any implementation of it.

- **Internal communication (importance increases when using iteration process):** because of its fast-paced nature, delays due to missing or incorrect data can cripple the team’s ability to perform. Co-location of personnel encourages informal communication; however, co-location is more dependent upon where people spend time than the office arrangements. Project-specific communication mechanisms such as forums can be used to filter, collect, and segregate information with project relevancy, as well as provide a way to ensure access to the same information by all members regardless of geographical location. One of the ways WSM system test and performance test could proceed in parallel and with the high rates of code change that iteration utilized was that everyone – designer, tester, coder – was often present during a hardware test shift. Thoughts about “interesting” test scenarios mentioned by the designers in response to observed behavior were recorded and/or immediately acted upon by the testers. Several times the testers made suggestions for significant
design changes that were in fact adopted, which is highly unusual in the traditional process implementation.

- **External communication**: communication with important parties outside of the team must be considered. In the case of WSM, the designers and SRM functional management kept other stakeholders informed, and the forum provided first line management access to the team’s day-to-day status. Often this communication occurs through informal relationships.

- **Centrality of build (critical to iteration process)**: with several groups testing code concurrently in different environments, the project motto must be “do not break the build”. Preferably, the same person should coordinate fix installation on every copy of the build, regardless of the functional group who “owns” the build. This role requires an experienced person who is proficient at tracking and coordination. The source code also must be carefully tracked to avoid losing fixes.

- **Tools support**: the testers wrote new tools that drastically improved test effectiveness, and used the feedback to suggest design changes. Functional management did not interfere with this effort, despite the existence of an “official tools group” within the laboratory. Where existing tools were modified, the tool maintainers were utilized.

- **Stable code base (critical to iteration process)**: iteration introduces code changes very rapidly. If the underlying code on which the iterative portion depends is not of high quality, iteration is impossible. The traditional component testing of stages one through three demonstrated the base code quality was high enough to enable algorithm testing at the system level, without which iteration would not have been possible. Each iteration’s code must also be of sufficient quality before building upon it.

- **Problem diagnosis (critical to iteration process)**: iterations contain code that undergoes less than normal testing before being made available to others. Anything that reduces defect injection (better than average coders, targeted inspections, CASE tools), speeds defect detection (automated regression tests, use of “null pointer” values which fail on read as well as write accesses), or speeds problem diagnosis (fast-fail modes, testing “hooks”, zaps) is more important when using iteration. Diagnosis is facilitated by more frequent iterations, since the amount of new code per iteration is smaller. When a problem occurs with a new iteration, the new code is presumed guilty until proven innocent. If iterations are done less frequently, more code is integrated and a more general (and hence slower) diagnosis process is usually necessary.

- **Fix turnaround time (critical to iteration process)**: Often when a system dump was taken during a test shift, it was debugged and sometimes fixed by the personnel present on the test shift even as the test continued. Ordinarily such dumps would be queued up to a central debugging group, who would then analyze the dump,
document the defect, and queue it to development for analysis and fix generation. Once a fix was coded and unit tested, it would go through several more queues before becoming available to the original group. This can take weeks of elapsed time. Fix response time will gate iteration cycle time; even for non-critical problems, with slow turnaround times time and effort will be wasted on duplicate problem discoveries.

- **Transition to a heavyweight project manager:** When the project slipped into a crisis situation, the resolution was to install an informal heavyweight project manager. The formal organization was not changed, but the functional managers were told to support the overall technical leader in very clear terms by the laboratory director.

- **Different processes and organizations for different parts of the same project:** The normal assumption is that the entire project must follow the same process. Sometimes however decomposing the project and following different processes makes sense: WSM had a portion that was independent of the underlying storage management algorithms, separate from the algorithms themselves. The former was well understood, low risk, and was required to test the algorithms; this part of the project was delivered using the traditional process (code stages 1-3). The algorithms however had a significantly different risk profile, and in the end, a very different process was required to develop them.

- **Matching process to risk profile:** whereas the waterfall process normally employed by this organization manages low to medium levels of product risk, the iteration process is better at managing products with high risk, because it generates knowledge with less elapsed time investment and feeds that knowledge back into the product before customer delivery. Barry Boehm, a noted software risk expert, comments on iterative processes: “…this book illustrates the new processes created by successful companies to address the risks involved in competing on Internet time” [Cusum1998a]. A Microsoft senior VP adds: “It appears that this incremental approach takes longer, but it almost never does, because it keeps you in close touch with where things really are” [Cusum1995a, pg. 202]. The ingrained belief within this organization that the waterfall process represents lower risk shows confusion between the attributes of the product and the process. It has used the waterfall process successfully for so long that if anything does go wrong, process deficiency is the last suspect.
Case Study: Workload Manager Release 1

Workload Manager Release 1 (WLM1) consisted of a set of new functions (features) added to an existing product (MVS). WLM1 included changes to an existing component (SRM) of MVS, as well as the addition of a new component (WLM). WLM would generate no direct incremental revenue, although it was expected to drive later sales by encouraging customers to embrace IBM’s clustering technology.

Background

Workload Manager was an important enabler in the MVS platform’s clustering (sysplex) strategy. MVS had introduced the operating system services necessary for integrating multiple systems into the appearance of a single larger system image, also known as a “system complex”, or sysplex in the release prior to WSM. Sysplex represented a solution to the problem of providing higher availability of computing services without requiring enhancements in fault-tolerant hardware or software technology. By replicating servers and providing a single system image, the outage of a single physical system would not make the service unavailable. This strategy assumed that installations would be able to add and manage the new servers with sub-linear support cost increases. Without a single system image, support costs would scale super-linearly. WLM1 was intended to provide a sysplex-wide scope to system performance management, traditionally a labor-intensive area of system management, by adding a new set of features to the MVS product to enable a single system image of performance management across a sysplex.

WLM1’s basic concepts paralleled those of WSM, but with an increase in scope and function. In addition to allocating main storage to work based on the observed behavior of the work, WLM1’s scope would also include managing CPU access and swapping, and allocation decisions would be based on all work in the system rather than the subset that WSM concerned itself with. The complexity of managing even a single system would also be reduced, by replacing the existing resource-centric performance specifications with human-centric specifications such as response time. Because of this larger scope, more personnel were required to develop WLM1 over a longer period than were required for WSM. Whereas WSM averaged seven to eight people with a peak of twelve, WLM averaged 21 with a peak near 35. Finally, those involved with WLM1 from the beginning were well aware of the organizational issues encountered by WSM.

Stakeholders

WLM was intended to simplify system management for every customer, even those who did not exploit the sysplex clustering technology. MVS performance management was recognized as a very labor-intensive process: for larger customers, WLM would allow exploitation of sysplex clustering for availability benefits without incommensurate scaling of system support costs. For smaller customers, it would allow highly skilled personnel to spend more time supporting the business rather than supporting MVS. Non-IBM suppliers of performance monitoring and management products (called Independent Software Vendors or ISVs) also had to pay close attention to WLM. Some companies
derived most or all of their revenue from monitoring products, which were intimately dependent upon MVS internals such as control block structure, and WLM would radically alter those structures. For competitive reasons, ISVs had to be sure that their products worked correctly when the new MVS release became available. Perhaps unintuitively, IBM was also highly motivated to have the ISVs ready – many customers considered the ISV products essential to their operations, and would not buy the new MVS release without being assured of compatibility.

As a strategic piece of the sysplex platform, visibility was high with IBM Marketing and upper management. Both groups realized based on customer feedback that WLM was an important piece of the value statement for the release, and without WLM a significant segment of customers would not be easily motivated to buy the new release.

The various functional areas – design, development, build, system test, performance test, publications – had all been touched by WSM, and feared a much larger-scale repeat. As with WSM, design, system test, and performance test were expected to be more heavily involved throughout the development of WLM than in most other projects. This is not surprising given that the same MVS component (SRM) would be involved, but the scope of changes were much larger and unlike WSM involved significant changes to (in many cases outright replacement of) external interfaces. Another factor contributing to the increased size of WLM relative to WSM was the single system image notion; whereas WSM testing always involved a single system, WLM testing would have to include multisystem sysplexes as well, greatly increasing the test setup complexity. High-ranking designers also had a personal interest in the success of WLM, having nurtured the idea for several years before it was committed to a release plan.

**Primary Risks Addressed by Project Plan**

Relative to the average project in this organization, WLM was extremely ambitious. Innovation risk was high for the central resource allocation algorithms, as the designers had only a high level idea of how certain functions could be implemented. There was no analog to the patent that formed the foundation of WSM, only some basic research and the designers' beliefs as world experts in MVS performance that it could be done. Implementation of the interfaces for specifying performance goals and presenting a single system image across a sysplex had relatively low innovation risk: a good deal of it consisted of familiar functions that had been implemented by the organization regularly in the past. Because there was some platform-level uncertainty about the preferred form of future user interfaces, the small subset of interfaces dealing specifically with the user specifying a performance policy carried the possibility of innovation risk early in the project. This uncertainty was resolved in a way that led to low innovation risk for this section of the project as well.

Schedules were considered extremely risky, for a number of reasons. First, there was the innovation problem: without the basic functions working, the release could not ship. The migration path for exploitation of the new goal-based performance specifications largely mitigated this risk. Second, unlike WSM, WLM dealt with a number of inter-product interfaces. Further, some of WLM's advanced functions required exploitation by IBM’s
hierarchical database manager (IMS) and transaction manager (CICS). IMS and CICS required interface code from WLM delivered on early MVS builds, so that they could make the required changes in CICS and IMS, test them, and deliver early versions of CICS and IMS back to WLM for testing of the related resource allocation algorithms. This schedule interlock required careful management, especially since CICS is developed in the United Kingdom and IMS in California, while WLM was developed in New York. Schedule slips in any of the three laboratories had the potential to affect the others. In addition to CICS and IMS, the team in Germany that owned IBM’s entry in the performance monitoring arena would also be affected by some externals changes, although this was a one-way rather than cyclic dependency.

Unstable requirements were also a risk for certain portions of the project. While the scope of the resource management algorithms was well understood and thought to be stable, and much of the single system image function was low risk, the user interface portion and certain key inter-product interfaces were expected to be vulnerable to requirements change. This contributed to the decisions to prototype the user interface code and to write the final version in a programming language that supports very rapid and easy modification compared to the language used for system code. In order to manage the risk of interface changes, especially with CICS and IMS where additional coordination was required, senior level programmers were assigned to define and manage those interfaces.

The process risk of using a CASE tool for code development was carried over based on the positive experience from WSM. Most new modules were written using the CASE tool, while existing modules were not converted to the CASE format. This risk was felt to be very low based on the WSM experience, and to further moderate it the two designers of the CASE tool (experienced MVS designers) became part of the externals team for one year.

**Product Plan**

The WLM product plan divided the project into three distinct pieces: the user interface, the algorithms, and the externals (cross-product interfaces, single system image, etc.). This cleavage reflected the product architecture, in that the resource management algorithms were nearly independent of the externals once a set of performance goals had been specified, and that the user interface for specifying that set of goals had very little impact on the rest of the externals. The externals and user interface portion were partitioned into a completely new MVS component (WLM), while the algorithms were implemented as a new subcomponent of SRM, much as WSM was implemented.

Delivery of external function was staged, to provide component test the ability to overlap testing of stage 1 with the development of stage 2, and so on in the organization’s traditional waterfall model. The content of early stages was defined largely by dependency arrangements with other laboratories and between the externals and algorithms portions of the project. Stages were integrated into MVS builds (similar to stages, but agreed to by multiple projects across the lab) which were then shipped to the remote sites on an agreed-to schedule. Delivery of the user interface code was merged
into the externals portion of the project once its prototyping had proceeded far enough to create a stable set of requirements.

Delivery of the algorithms was staged to reduce dependencies between the groups, allowing them to proceed as independently as possible from each other, so that project management complexity was reduced. Scaffolding was created to allow specification of performance goals during algorithms testing, which was used until much of the externals code was written and stabilized. As large pieces of the algorithms became stable, they were integrated into the common base but otherwise remained outside of the normal code library management system.

**Process Plan**

The product partitioning into externals, user interface, and algorithms was mirrored in the process plan. The externals were developed via the traditional process, the user interface underwent iterative prototyping followed by the traditional waterfall process for the final implementation, and the algorithms were implemented via a modified version of the WSM iteration process, so that the maximum rate of learning would be generated, reducing downstream risk.

One of the consequences of WLM’s larger size relative to WSM was that the iteration process needed to be scaled up. Whereas a single person wrote code for most WSM iterations, six people wrote WLM algorithm code. Modifications to the process were made in a collaborative fashion; the primary changes were a reduction in the iteration frequency due to the increased volume of code required for the additional functions and coordination overhead, and the addition of a “smoke test” simulation for large or complex changes prior to integration into an iteration. Even such details as the day of the week when an iteration would be integrated were carefully examined to ensure that the resulting build would be operational as quickly as possible even if problems were found in it during the build process.

A second consequence was the necessity to formalize the role of iterations and early algorithm testing, in order to make it more acceptable to the project management structure within which WLM was situated. The iteration process was documented and ISO9000-certified, the algorithms testing process was christened Algorithms Verification Test, and both were integrated with the overall project test plan. This formalization, and somewhat the legitimization conferred by ISO certification, greatly reduced the resistance of functional management to the process.

WLM’s larger size also required more attention to internal communications. Multiple electronic forums were set up, one of which was the official project-wide repository for decisions. Thus team members could be informed (or inform others) of any change simply by using this forum. Minutes of critical meetings were posted so that even distributed members of the team (management, Germany) could remain informed in real time of decisions that might affect them. A concession to both communication and coordination was the advent of several regularly scheduled meetings. The externals and algorithms teams each had distinct weekly meetings to discuss staging, dependencies, and
in the case of the algorithms team which tests (a) could be run on the current code (b) the best order to run those tests in order to maximize learning. In addition, a project-level meeting was held at which all decisions about external interfaces and behaviors were made. The senior members of each project group both ensured overall architectural integrity and provided technical input to the decision-making process. In a similar way, the project test plan was integrated to maximize coverage and minimize duplication from algorithms verification test through performance test, rather than observing the normal functional organization limit by excluding system test and performance test.

Mechanisms were also put in place so that the overall project architect and important members of the externals group had regular contact with customers under confidentiality agreements to critique the current design and act as a sounding board for decisions. This feedback consisted of a mix of one-time reviews by customers new to WLM and a handful of customers involved over a longer term who provided a more in-depth perspective.

![Diagram of project development stages](image)

**Figure 14: Workload Manager Release 1 Schedule Overview**

**Organization Plan**

WLM’s formal organizational structure fit much better with the project than WSM had. Many of the WSM designers were involved with WLM1 while it was a research project, so they were acutely aware of WSM’s problems and structured the WLM1 project organization to address some of these problems. Almost the entire WSM algorithms team, and some of the development group, moved directly from WSM to WLM1. The chief architect of WLM1, although not a veteran of WSM, was a senior member of the technical community (one rung below IBM Fellow, IBM’s highest technical position) that had significant organizational influence. He had extensive contacts with upper management and executives, which he used to influence the management of functional areas WLM was dependent upon, making him a virtual heavyweight manager. He also paid close attention to the “soft” (human) aspects of the project, preferring to allow project engineers decide process issues rather than automatically following the prescribed process. WLM’s larger size relative to WSM meant that two first line departments would
likely be needed, based on organization norms for department size. The division of labor between the two departments was along product and process lines: algorithms in one department, externals and user interface in the other.

Initially the algorithms department reported to the MVS Design organization and the externals/user interface to the MVS Development organization, mirroring the structure of WSM. Once more than a handful of project staff began to arrive, the algorithms department was moved into the MVS Development organization so that both departments reported to the same second line manager, reducing the organizational dispersion of the project personnel. All of the designers, coders, and component testers reported to one of these two departments. Coordination mechanisms were constructed as described previously, but much of the day-to-day work of the departments was done independently. There was more internal cohesion within departments than across them.

![Organization Chart]

**Figure 15: Workload Manager Release 1 Organization Overview, after reorganization**

A great deal of attention was paid to staffing as well. Because of the stringent requirements for code quality in order to enable an iterative process for algorithm development, a small group of very experienced people was assembled to design and develop the algorithms as well as manage the iteration process. Most of these people were experienced not only in MVS, but either in SRM itself or in components just on the other side of major inter-component interfaces from SRM. Even a first line functional manager was chosen based on his experience in managing a closely related area, an MVS performance test group. The development team leader for the externals group was an experienced SRM developer and veteran of WSM, who understood both the traditional and iterative processes. Expertise in the CASE tool was assigned to the team as well, as were experienced designers from other parts of the lab. In many cases the more senior members of the two teams had worked together previously on other projects, which reduced the initial storming [Tuckm1965a] associated with new team formation.

Because of the paradigm shift required by the new system performance management externals, the unusual step of assigning a technical publication writer to the project
occurred more than two years before the product was released to manufacturing. This person was considered part of the development team (although she still reported to a different functional area), and attended meetings oriented towards externals as the decisions affecting them were made. In the tradition of WSM, the system test and performance test groups were represented in the algorithms team meetings by people from those functional areas dedicated to WLM. They provided access to hardware environments for algorithm verification testing, and helped to define which tests were run. This enabled the functional areas to gain competence with WLM prior to the code being integrated for their independent tests, as well as allowing them to understand exactly what types of testing had been covered by other areas.

Finally, there were a number of active ambassadors and gatekeepers, both formal and informal, managing critical interface points between the two WLM groups, briefing customers, and keeping management informed of progress. Bringing the other functional areas into the team meetings allowed their representatives to filter information and communicate the functionally relevant portions back to their organizations.

**Project Evolution**

From an overall IBM perspective, WLM was clearly a success. The iterative algorithm process allowed all of the resource management algorithms considered candidates in the original plan to be finished without moving out the MVS release date, although some final modifications were tested in parallel with initial manufacturing and delivered as maintenance. The related IBM products – performance monitor, database, transaction manager – all had the required code, and early versions had been used to verify that the resource management algorithms exhibited the desired emergent behavior. ISV products were ready concurrently with MVS release availability to customers, and customers who considered the current state of system performance management inadequate began migrating. The Algorithm Verification Test addition to the formal process, coupled with active organizational boundary management, reduced the organizational tensions to normal levels. The overall project organization remained essentially intact throughout the project, although some fine-tuning occurred along the way.

Interactions with outside organizations proved problematic at times, even with explicit boundary management. The IMS (database) and CICS (transaction manager) organizations were on different product schedules than MVS, and the formal mechanisms in place to manage cross-laboratory dependencies occasionally broke down and problems needed to be resolved via informal means. Schedule delays in the availability of stable test versions of these products, which contained the WLM-specific modifications, would have impacted WLM schedules, but delays in WLM staffing outweighed these external delays. The staffing delays reflected the project leader's uncompromising demands for some of the best-in-house talent, rather than settling for average personnel in critical positions, but also had effects on other organizations. Cooperating laboratories interpreted the staffing delays as a lack of commitment to the project, requiring effort on the part of ambassadors to correct. Once the staffing problems were solved and significant amounts of function were made available by the externals team, the external
problems abated and the demand for component testers outstripped supply. Many short-term on-loan personnel were needed to keep the test effort on schedule.

Test was a vulnerable aspect of the project, because of its complexity and scope. There was no local expertise in testing with CICS and IMS, so the algorithms verification test leader had to develop that expertise and install the products on test systems. Another project in the laboratory in the same MVS release, also critical (arguably more so) to the sysplex strategy had consumed much of the senior component test resources. WLM managed to recruit a senior tester as the externals test team leader, but she had limited experience with SRM. One of the WSM test team leads, logical choices to plug this hole, had moved on to another laboratory. The other was responsible for some algorithm design and code, as well as iteration builds and the related process tools support. As test began to fall behind, this latter person was made the overall project test leader while maintaining build duties, which were recognized as critical to the algorithms' progress.

As the project progressed and other personnel in the laboratory became available, this job became focused strictly on managing algorithms test and code integration across the various test groups (three to four separate copies of the iterations were maintained in parallel for different test environments). A senior tester with SRM experience returned from a leave of absence and took over many of the administrative aspects of managing and coordinating the overall test effort; an experienced system-level debugger became available and assumed ownership of problem diagnosis.

As part of the scaling up of the iteration process, two process-specific metrics were formulated to provide an objective measure of progress. The first of these was "code velocity", defined as the sum of the number of new, changed, and deleted lines of code (LOC) introduced in an iteration. Since testing could result in the replacement of large amounts of code compared to the traditional specification-centric process, it was important to capture deleted LOC; traditional process measures counted new and changed code, but ignored the amount deleted. The intuition was that as the algorithms for a particular resource converged, changes would be smaller and code velocity would be reduced. If deleted LOC were ignored, adding 2,000 LOC to one function and removing 2,000 deemed not working from another would yield 0 new/changed LOC due to aggregation of the measurements, rather than the 4,000 LOC actually changed. The second new metric was progress against a set of ten customer-oriented scenarios representing different "problems" that the algorithms needed to solve. Cases ranged from relatively simple single-workload to complex mixed-workload with resource contention. Progress was qualitative, with values to distinguish three cases:

1. Waiting for enough algorithm functionality to be able to run the test scenario.
2. Scenario run, results analyzed, and changes deemed necessary.
3. Scenario run, results analyzed, and designers satisfied with results.

Part of the results analysis included traditional performance test measurements, so success against these criteria made it very unlikely that further functional modifications would be necessary before release to manufacturing.
The biggest single problem turned out to be a subcomponent which did not cleave cleanly along the lines used to partition the rest of the project. Considered to be well understood, its design was deferred as the over-loaded externals group concentrated on other problems. It was staffed relatively late in the project by a developer with no prior SRM experience, required modifications to modules owned by both groups, had stringent performance and run-time environment requirements, and involved inter-product interfaces that were being re-architected. Personalities and incentives also entered the equation here. The developer of this subcomponent had personality clashes with several members of the project, including people in the cooperating product interface areas, which temporarily damaged the working relationship between the groups. Completion of the subcomponent was also holding up this employee's transfer to another site, so there was a considerable short-term incentive to finish as quickly as possible. Based on defect data recorded over the next few years, this subcomponent has the worst quality reputation within WLM.

**Discussion**

If WSM demonstrated the dangers of a mismatch between product, process, and organization, WLM demonstrates just as surely how positive the confluence of these factors can be. By observing the personal lessons from WSM, those involved with the design of the WLM project found ways to legitimize the iterative process within the organization and to match the organization more closely to the project. Therefore, a much larger, much riskier, and much more complex project with more stakeholders went much more smoothly than WSM. Attention was given from the beginning to the organizational structure, including conflict resolution mechanisms such as regular project meetings and clear areas of expert authority. As a technical judge of last resort, the overall project leader was available for consultation and would quickly settle disputes if overall progress was endangered. This person had the technical and business clout necessary to make things stick even in the face of some managerial resistance. Both technical leaders and management engaged in shielding of the team from outside distractions.

Algorithm verification test is still used by WLM for all new projects, a recognition by the organization of its effectiveness in reducing defects found in system test, performance test, and by customers. Algorithm designers in the development group work directly with the testers to define scenarios and the expected behaviors; no additional detail is required in the product specification, and no formal completion criteria is typically demanded by the release manager. The designers decide when the algorithms are working to their satisfaction. Given the early start, and short cycle times of this testing, since WLM there has never been serious danger of missing release code cutoff dates. Formal iteration is actually not used very often, for several reasons. First, the experience gained from WSM and WLM reduced the perceived risk of the mechanisms underlying the algorithms, so further enhancements are perceived to be incremental in nature. Second, several of the core WLM algorithms developers have remained with the component, further reducing the perceived risk of algorithms changes. Third, the acceptance of algorithm verification test in the formal process affords developers the opportunity to do small, informal iteration-like tests without the overhead of another process.
Lessons

As a large project in comparison to WSM, WLM of necessity dealt with a different range of issues in project design. Communication amongst the project members became more important, due to the larger number of people involved, the larger project scope, and the physical and organizational dispersion of cooperating groups necessary for delivery of the integrated solution.

- **External communication:** due to such factors as physical dispersion, organizational scope of the project, and heavy reliance on customer validation of the design, the gatekeeper/ambassador communication functions were quite important. The presence and conscious assignment of gatekeepers to critical interfaces enabled continued successful engagement with a large number of stakeholders.

- **Internal communication:** while the gatekeeping functions concentrate on filtering and mediation of outside ideas, just as crucial was the internal sharing of information for coordination purposes. Project forums again proved useful here, acting as newsgroups for distinct project constituencies and providing a means for geographically distributed team members to maintain first-hand access to real-time project decisions, status, and problems.

- **Iteration:** iteration is not only for very small teams, but the iteration process must be scaled up carefully. Because of its intensely focused nature, the iterative process demands shorter time constants from its supporting tools and processes, even for large projects. WLM executed 28 iterations over 19 months; as the number of coders contributing to each iteration increases, the effective iteration interval increases due to coordination problems, suggesting a rather small group size as the practical maximum for an iterative process.

- **Alignment of process and measurements:** new processes often require new metrics. The creation of the code velocity metric and set of test scenarios that acted as completion criteria for the algorithms satisfied the both organization’s need for interim measures of progress and the project’s recognition that the traditional process measures of defects and person-months made little sense in the context of iteration. Iteration relies less on pre-integration testing, so more defects are expected given a constant injection rate (the traditional measure only counted post-integration defects discovered). Likewise, the iteration process is used precisely when uncertainty about the correct solution is high, so more rework is expected. The point of iteration is to compress the elapsed time of the process by using early learning to reduce downstream rework.

- **Tools:** the larger scale of WLM required some modifications to tools used for development and code integration, which the team handled. As before, the normal support group was used for the CASE development tool. While the test tool from WSM was carried over, WLM’s larger scope meant that the real-time interface did not scale well. This portion was scrapped, and a post-processor module added, again
by the development group.

- **Heavyweight project management:** The formal organization was strictly functional, but the project chief architect wielded considerable influence through boundary management. He had, and used, contacts with upper management and executives to exert influence on functional management.

- **Different processes and organizations for different sub-projects:** As with WSM, the overall project was cleaved along its natural risk profiles, and the pieces allowed to develop independent risk mitigation approaches. In spite of the scale-up along several dimensions, WLM as a whole proceeded much more smoothly. Perhaps most telling, the subcomponent with the poorest fit to this project structure was the source of persistent difficulties. This was a functional component, which, while predominantly contained within the externals area, needed significant co-requisite changes in the algorithms component.
Case Study: Workload Manager Release 4

Workload Manager Release 4 (WLM4) consisted of a set of new functions (features) added to existing components (SRM, WLM) of an existing product (OS/390). WLM4 alone would generate no direct incremental revenue, but its enhancements for processing batch workloads coupled with recent hardware announcements was expected to regain market share lost to Hitachi’s Skyline processors.

Background

Workload Manager release 4 (WLM4) was the fourth stage of WLM’s evolution, included as a part of OS/390 Release 4 (R4). Like WLM1, WLM4 was a set of features added to MVS (now part of the OS/390 product) rather than a new component or product. Work started on WLM4 3.5 years after WLM1 ended. During the intervening time, two stages of WLM evolution had added transaction management facilities, and given the WLM component the ability to start and stop servers for a given workload within a single system. The S/390 hardware market had been completely transformed: IBM had changed to CMOS technology for all mainframes even though CMOS uniprocessor speeds were lower than unprocessors built using bipolar technology. This provided a significant market opportunity to Hitachi, who continued to manufacture bipolar machines in addition to introducing models based on CMOS technology. With essentially no bipolar competition, Hitachi was able to charge a premium to customers whose workload was sensitive to uniprocessor speed.

WLM4’s content was a hodgepodge of seven items addressing four different market need areas that varied greatly in almost every relevant aspect, as is common in a mature operating system component. These are summarized in the table below.

<table>
<thead>
<tr>
<th>Table 5: Workload Manager Release 4 Functional Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Market Area</strong></td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>Resource</td>
</tr>
<tr>
<td>management</td>
</tr>
<tr>
<td>Web server</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Base</td>
</tr>
<tr>
<td>enhancements</td>
</tr>
<tr>
<td>Batch</td>
</tr>
<tr>
<td>management</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

From a marketing perspective, only the batch management area would be important enough to receive special attention in the product release announcement. These two items were booked very late in the release plan, based on marketing considerations. Work on them started too late in the release development cycle to be complete in time for
the release code cutoff milestone, so they were given special handling from the start. The total package of code required to satisfy the requirements included changes to WLM, JES2 (job entry), SDSF (job display), and RMF (performance monitoring) elements of OS/390.

WLM4 was intended to do three things: support the Network Computing initiative by providing web server support, clean up some of the more visible warts in WLM, and attack “the batch problem”. Every OS/390 customer runs batch work, and has to contend with a number of problems in managing that workload. From a customer standpoint, some of the problems currently solved by add-on software packages would now be integrated into the operating system. Affinity scheduling provided universal availability of these functions, without additional expense beyond the base operating system cost. Since IBM’s sysplex strategy increased the likelihood of certain problems related to software licensing, solving this in the base operating system supported the sysplex strategy. Initiator management provided workload balancing across a sysplex, long sought after by installations running multiple operating system images. Initiator management supported the sysplex strategy by making the transition to multiple images easier for smaller customers. All of this was carefully positioned with the sellers of ISV batch management products, to reassure them that IBM was not pursuing their market.

**Stakeholders**

Once educated, most ISVs considered batch management a boon; since IBM was providing official support for some functions ISVs performed using undocumented interfaces, their support costs would be reduced and their products would still be attractive to customers based on other functions. As with WLM stage 1, IBM was also highly motivated to have the ISVs ready with their co-requisite support at release availability – many customers considered the ISV products essential to their operations, and would not migrate to the new release without being assured of compatibility.

As a strategic piece of the sysplex platform, batch management was highly visible to IBM Marketing and upper management. Both groups realized based on customer feedback that batch management was an important piece of the value statement for the release, and without it a significant segment of customers would not be easily motivated to buy the new release.

The various functional areas – design, development, build, system test, performance test, publications – all would be affected. The changes in areas other than batch management were small, booked in the normal release schedule, and their implementations were well understood, so these items were considered mostly traditional. Batch management would require a large number of changes to the JES2 as well as OS/390 publications, and the testing of JES2, SDSF, and WLM support would require high coordination effort and specialized planning. Since both JES2 and WLM managed sysplex-wide resources, testing would have to include mixed levels of both products, adding to test complexity.
Primary Risks Addressed by Project Plan

The five items not related to batch management were thought to be very low risk. Three of these required only the WLM department and normal downstream processes such as system test. Each web server item involved one other department, one in Poughkeepsie and one in Raleigh, NC, but they were small incremental changes to existing functions. All five were well understood, with an emphasis on extension of existing designs rather than invention of new ones. No special risk management techniques were considered necessary for these items.

Batch management was considered very high risk, primarily to the release schedule, and its addition to the release plan was made over the strong objections of the release manager. Since code delivery could not be completed prior to code cutoff for the release, special internal delivery dates were negotiated. Innovation risk was moderate in the algorithms for starting batch servers and low in the affinity scheduling area. Initiator management entailed the use of data not used by the previously delivered server management support, and had a multisystem rather than single system management scope. As part of this scope change, a peer-peer algorithm for balancing the servers across systems had to be invented. The only cross-site dependencies were with IBM's performance monitoring product, which had been routinely managing WLM support since WLM1. The changes for WLM4 were minor, and the WLM lease team leader managed the interface with this outside group. There was some perceived risk in the delivery of the JES2 portion of the code, but it was assumed that this would be resolved by increased management attention to this group.

Product Plan

The initial product plan dealt only with the items outside of batch management, since batch was added to the release plan late in the development cycle. While development expected its addition to the release plan, only limited headcount could be diverted to it while it was out of plan. The emphasis was on early completion of the small items, so that development personnel would be available to start work on batch management if it was added to the release plan. There were relatively few intersections between the items, so they were well underway by the time batch management was formally added to the plan.

Batch management was a highly collaborative design, whose content had been repeatedly refined with customers presentations over several years. The WLM architect secured funding to hire an IBM Global Services consultant for one year to help define the specification, review it with customers, and make sure that it was coherent from a customer perspective. This consultant and some colleagues were experts on batch issues, and in addition to the specification they published informal question and answer documents for customers under non-disclosure agreements, which became the basis for arriving at a shared understanding of the product. Even before WLM was ready to ramp up its development effort, design meetings were held where both the WLM and JES2 designers discussed and resolved interface and design issues. Both WLM and JES2 needed to implement a new format for their sysplex-wide persistent storage as part of this support, which had design and test ramifications. SDSF, which included batch job
monitoring functions and was heavily dependent upon JES2 internal structures, interfaced primarily with the JES2 group.

**Process Plan**
For non-batch items, the traditional process was to be used. In cases such as the web server items, where for setup reasons it saved effort to have the exploiting components test the WLM changes, this was done with the consultation of WLM testers. The local PI evaluation and policy activation items were primarily performance items, and algorithms verification testing was used to validate the results. Independent system test would conduct its normal stress testing, but otherwise was largely unconcerned with non-batch items.

Batch management was simply not testable on the release schedule using normal system test processes. System test was already over-committed prior to the addition of batch management to the plan. Because the affinity scheduling item was well-understood (low innovation and requirements risk), the normal waterfall process was used. Initiator management, on the other hand, included new algorithms with higher innovation and schedule risks; this item was developed iteratively. Plans were negotiated that allowed system test of batch management to continue past the normal release end date, and some of the normal system testing was performed as part of algorithms verification testing for the WLM items. There was extensive coordination between WLM and JES2 component test, system test, and algorithms verification test.

![Diagram of Workload Manager Release 4 Schedule Overview](image)

**Figure 16: Workload Manager Release 4 Schedule Overview**

**Organization Plan**
Much like WSM, this project was dropped into the existing organizational structure and the various sides told to "work together". For the five items with relatively small outside interactions, this was expected to be quite adequate. The WLM development group was quite experienced: of the nineteen full-time employees in the department, thirteen had worked on WLM continuously since stage one and only three had never worked on WLM before. All of the designers and code-- were amongst the veterans.

The chief architect attempted to have the JES2 group moved under WLM management during the batch management portion of the project, but this was rejected by upper management. This area of IBM rarely re-organizes around projects, except for the initial
release of a large new component. By then-current standards, batch management was
neither large nor new, which may have led management not to seriously consider this
option. The WLM architect, the same since WLM1, understood that leaving the WLM
and JES2 groups under separate first and second line managers was a risk. Like WLM,
the core of the JES2 development group had worked on JES2 for years. One former
member of the JES2 group had moved to the WLM group during WLM stage 3; this
person became the primary ambassador between the two groups, and an especially
valuable one since he could translate the component-specific vocabulary used by both
sides.

![Image of a diagram showing the Workload Manager Release 4 Organization Overview]

**Figure 17: Workload Manager Release 4 Organization Overview**

**Project Evolution**

As might be expected from the preceding, WLM4 was essentially two separate projects.
The five non-batch items were scheduled early to get them out of the way, freeing up
personnel for the anticipated addition of batch management and reducing project
management complexity going forward. The WLM release team leader was also one of
only two experts on SRM component design; this activity became the overall project
bottleneck, since all items other than web server support required SRM design work. Of
the five non-batch items, only the DNS support deviated significantly from planned
progress. A major process risk had been overlooked; the item required the addition of C
language interfaces for several existing services. The Poughkeepsie laboratory, with the
exception of the Unix Services group, was largely unfamiliar with C (an internal
programming language is normally used), and the DNS product was written only in C, so
the two sides of the interface had a great deal of trouble testing and debugging across the
language interface. Vendors (contract programmers) were used to test several of these
small items. For most of the vendors, this project was their first experience component
testing OS/390, leading to lower than expected productivity. Helping and managing them
required substantially more of the experienced component test team leader’s time than
had been budgeted, so testing duties had to be redistributed.

Batch management itself had two very different faces. Affinity scheduling was well
understood, easily partitionable between WLM and JES2, and relatively independent of
SRM. Since it was also by far the largest piece of work owned by WLM in the release,
designers familiar with WLM began work on this using the waterfall process while the
smaller SRM items were being finished. Not surprisingly, it also finished earlier,
enabling these developers to help test and provide fixes for the on-going test effort.
Initiator management, on the other hand, required the addition of a JES2-SRM interface with strict performance requirements and decisions about how various batch job states would be reflected in the data moving across this interface. Initial involvement from the SRM designers was time-limited to settling these questions. Their experience was that once the incoming data was well-understood and fit the general pattern used in the past, the initiator management algorithms could be written iteratively to make valid decisions based on the data. As with previous WLM projects, code delivery was scheduled to minimize the schedule risk of cross-component dependencies and to provide complete packages of function incrementally to allow parallelization of testing. Less complex sections of the code were handed off to personnel as they became available, but the SRM design time constraint clearly gated progress. As the product announce date came and passed, the initial low level design for the server balancing algorithm was just being completed; ordinarily, its implementation and component test should have been completed at this point in the release.

Due to the significant amount of testing still to go at the announce date, no mention was made of the batch management support in the OS/390 R4 announcement. Checkpoints were taken between management and the different internal stakeholders, especially the testers, to gauge readiness for announce. In the end, batch management was announced two months after OS/390 R4 was announced, along with the G4 mainframe models. Improvements to the algorithms continued for another three months before the designers were satisfied. This substantially matched the planned schedule shown previously.

**Discussion**

WLM4 blended many aspects of WSM and WLM1, providing a snapshot of issues at a later stage of product (in this case, the WLM component) maturity. Changes in each release were smaller, more numerous, and more incremental in nature, supporting a number of different company strategies. In addition to the technical aspects, the organization itself was experimenting with a cultural change by using contract programmers for certain aspects of the project. Consequently, project management complexity was greater than would be expected given the project’s size and some focus was lost. The loss of focus allowed a number of small risks to accumulate until certain parts of the project were heavily impacted. Skill bottlenecks began to appear as experts were forced to switch between disparate sub-projects and risk mitigation efforts.

Into this mix, business strategy dictated the addition of a comparatively large and complex change that required a great deal of interaction beyond the group’s traditional boundaries. While there was enough technical continuity to anticipate the impact of the organizational structure on the project, there was no corresponding management continuity. Without the WLM1 and WSM experiences to draw on, upper management was unconvinced of the need for a project-oriented organizational structure, even on a temporary basis.
Lessons

WLM4 had a wide range of internal contrasts amongst the various release items, which stressed the project management infrastructure. Two large high-risk items were focused on, which allowed several smaller items with unique risks to escape control for a time. Earlier lessons of organizational interaction were lost due to a lack of management continuity, allowing a repeat of past mistakes. Technical continuity was sufficient, however, so that the process lessons specific to iteration, tools support, and internal communication were carried over. Since no major modifications were added to those earlier lessons based on this project, they are omitted below.

- **External communication:** again, gatekeepers and ambassadors played a significant role in this project. The WLM architect worked with the project selection process to articulate the customer value of batch management, which resulted in its late addition to the plan. He also went outside the normal range of staffing, by securing funding from outside the development organization to hire a batch expert from IBM Global Services as a consultant. One of the WLM developers came to WLM from JES2 during WLM stage 3, and he provided a crucial informal link between the two groups. The WLM and JES2 department managers did significant work to negotiate delivery dates with the release manager and owners of downstream processes. The WLM release team leader managed inter-product interface changes for batch that fell outside of the Poughkeepsie site.

- **Contract programmers:** Brooks would be proud to see that the lesson which provided the name for his classic text on project management [Brook1975a, chapter 2] has not tarnished with age. While upper and middle management saw adequate staffing for the project based on headcount, first line management and the component test team leader were acutely aware that headcount figures overstated the productivity potential of the group. Steep learning curves experienced by the contract programmers forced the experienced component test team leader to drop all of her technical assignments in order to keep this group working effectively. Once trained, the best could be retained for later projects, but the initial training cost was greatly underestimated. Time spent training those not retained was lost.

- **Different processes for different sub-projects:** Different parts of the project, cleaved along planes of risk, followed slightly different processes. The core initiator management algorithms used a scaled-down version (fewer people, less code) of the iteration process, while the rest of the project strictly followed the traditional process. Performance-related items all underwent the SRM-specific phase of algorithms verification testing as well. Although certain phases were carried out by other groups (component test of DNS support was primarily in Raleigh), the basic process was intact.

The diversity of release items suggests a refinement of the general warning against a mismatch between product structure and organizational structure. Problems did not result from the crossing of organizational barriers when developing the fork performance item, but significant problems occurred in both DNS and batch
management. The distinguishing factor appears to be the presence of other significant risk factors for the item.

The fork performance item was absolutely routine save for the exploitation by a group outside of WLM, and the two departments worked together to own the parts of the process for this item that made the most effective use of overall resources. DNS suffered from the differences in culture and technical expertise between Poughkeepsie and Raleigh; crossing the programming language barrier proved harder than either side expected, and there were problems between the two groups in understanding exactly what level of testing each location would perform. Batch management had complex tightly coupled pieces, crossed an organizational boundary with significant microcultural differences, and was more organizationally dispersed than WSM. The distinguishing difference between the fork performance item, which worked well, and the other two, which did not, was the presence of other significant risk factors in addition to a mismatch with organizational structure.
Case Study: Component Broker for S/390

Component Broker for S/390 (CB/390) was a new middleware product for the OS/390 platform. It was the S/390-specific implementation of the CB solution family, designed to provide enterprise-wide CORBA object support.

Background

During the 1990's, object-oriented programming techniques came into the mainstream. As businesses began to exploit this technology, common problems emerged with the existing implementations. These revolved around issues of interoperability between different object-oriented environments, security, scalability, performance, and the difficulties in presenting an object-oriented interface to data stored outside of the object-oriented environment. When the data required to instantiate an object originated in multiple places (databases, files, etc.) the problem of data consistency and logical object recovery across multiple storage products arose as well. With the exception of interoperability, the problems described above were all addressed by historical S/390 platform strengths.

IBM's first major offering in this area was the System Object Model (SOM), announced in 1992 for OS/2 and AIX (IBM's brand of Unix) and later ported to MVS. In 1994, IBM announced its next generation of object server, the Distributed System Object Model (DSOM), which implemented CORBA compliant interfaces. Still aiming at a moving target, DSOM for S/390 was delivered for general availability in late 1996, several months after IBM had announced that DSOM support would be dropped in favor of the next generation, Component Broker. Component Broker was a new, separately priced and orderable product with several platform-specific implementations.

The Component Broker S/390 (CB/390) solution provided a CORBA 2.0 compliant, secure, and transactional execution environment for the deployment of distributed business objects on the OS/390 operating system. Many of the lower level object services defined by the Object Management Group (OMG) were integrated into the server runtime, freeing business application programmers from explicit knowledge of and direct use of the low level services. The intent was to allow application programmers to concentrate on solving the business’s problems rather than building server infrastructure. The Component Broker S/390 Server was built upon a CORBA 2.0 Internet Inter-ORB Protocol (IIOP) compliant object request broker, allowing the server to interoperate with other CORBA compliant object request brokers in a heterogeneous and distributed network. The CB/390 implementation was intended to apply OS/390's traditional strengths (scalability, security, performance management, etc.) to the object-oriented arena, without the requirement to move or replicate the underlying data to other platforms.

The DSOM implementation on S/390 provided an opportunity for this organization to understand the mechanics of porting industry standard object-oriented technology in detail, but failed to provide experience at designing new function within the standards.
Although DSOM was predominantly a port of code from other IBM platforms, the
development group was staffed with some of the leading OS/390 developers. During the
porting process, they implemented platform-specific enhancements to improve the
reliability, serviceability, availability, and performance of the ported code. They also
encountered significant process problems during code integration, and were forced to
solve them in an ad-hoc fashion.

DSOM’s development group had felt a strong need to reduce their exposure to disruptive
elements. Technology, programming language, and integration elements were dictated
by external decisions, however build and development library support was missing.
Because of unacceptable technical limitations in the alternatives, the team created
specialized tooling that automated and parallelized the build process. Due to the
expected availability of library system support, the team chose a manual library system
approach rather than implementing something that would need to be reworked in six
months. The OS/390 organization, used to tightly controlled build processes, viewed this
as ad-hoc even though the group developed specific tooling to limit the risks involved.
The development team exercised a significant amount of control over the project, and
used that to reduce risk by making very careful tradeoffs between function and stability.
Experience from SOM R2 had motivated the SOM R3 team to put hooks into the code
that allowed some testing to be performed in a more productive environment.

**Stakeholders**

There was a wider array of stakeholders involved with CB/390 than most other projects
undertaken by this organization. In addition to the various functional areas within the
laboratory, other platforms were heavily involved with the overall IBM Component
Broker delivery. An IBM-wide cross-functional team was created to coordinate the
delivery of the Component Broker Series; the lightweight project manager for CB/390
was one member of this team, rather than the driving force behind it as would typically
be the case. Because of market timing issues, there was heavy pressure on this team to
deliver the product according to a schedule set before a detailed technical assessment was
possible.

The S/390 Software Design Council (SDC), a group of senior architects from each of the
major IBM products involved with the S/390 platform, had spent considerable time and
effort on CB/390. The lead architects for CB/390 were also members of the SDC, and
had worked on DSOM as well. This group’s primary concern is technical performance:
providing the classic S/390 strengths and ensuring that the product satisfies customer
requirements.

Customers wishing to exploit object-oriented technologies were of course eager to begin
testing with IBM’s S/390 implementation. An implementation with S/390’s classic
strengths would provide them with low risk alternative to leaping into unfamiliar
technologies on unfamiliar or less robust platforms. It would also provide them with
functions, such as transactional data coherence across multiple databases, that they would
otherwise be required to implement themselves. Finally, customers would be able to
access legacy data without the complexity of database copies and replication.
The various functional areas were not all quite sure what to make of CB/390. For those who had not worked on DSOM, it was completely different from any project in the organization. The pressure from the overall CB schedule trickled down onto these people, many of whom had to simultaneously learn new technologies and deliver function on a tight schedule. The project's use of C++ and industry standard technologies was an attraction to some; since IBM's layoffs in 1993, more attention was paid to maintaining skills that would keep one employable elsewhere. The bulk of mainframe code is written in a proprietary IBM language, which has little value in the external job market. The ability to cite work in C++ on a CORBA implementation would give a significant boost to one's resume.

Upper management (VP of S/390 software and above) had been convinced by the architects of CB/390's importance. Functional middle management, which had the responsibility for implementation, was not accustomed to projects with this degree of uncertainty. Their performance ratings were based on delivering projects whose functional content and timing had been committed to by the executives, whereas upper management understood that, beyond a minimum feature set, schedule was of greater importance than function delivered.

**Primary Risks Addressed by Project Plan**

The primary risk management strategies for this project were: re-use DSOM staff on CB/390, giving control of project direction to members of the OS/390 design group, updating the project size and schedule estimates after some experience had been gained, and using state-of-the-art tools for code development and project management. These actions were intended to mitigate several risks. Staff from DSOM project were experienced with C++ and objects relative to the rest of the programming laboratory, and with some of the development tools, reducing skill risk. Project control was given to architects from the platform design group because of the perceived requirements risk; they, in turn, had been clear in emphasizing the importance of customer feedback from an early beta test program as part of the planning process. This control was lightweight, however, since functional managers retained control of developers and testers working on the project. The project size and schedule estimates were reviewed after several months of work to see if their underlying assumptions, based on the DSOM project, were correct. There were concerns about the amount of savings that would be realized by exploiting existing OS/390 services rather than implementing them from scratch as the other platforms had done, and whether or not the assumed code complexity was correct. The decision to use leading-edge development and project management tools was not purely a risk management decision; there were issues of skill-building, proving out C++ tools used at other sites on OS/390, and so on. This decision did have the effect of altering the risk profile of the project, however. While it lowered process risk by solving problems such as library management, it created process and skill risks. The library management system used for most OS/390 code was incompatible with the C++ language format, which necessitated this compromise.
Outside the scope of the project plan were other risks. The architects acting as design leads on CB/390 attempted to address these independently of the project plan, but they bear noting because they are part of the contextual tapestry. Foremost among these risks was the worry that IBM management’s commitment to the overall Component Broker product set might falter, in spite of the architects’ belief that a market presence in objects was critical to IBM. Although it was believed that commitment within the S/390 management chain was secure, other divisions with less certain commitment levels were involved. Thus, a portion of the architects’ time was spent maintaining and nurturing commitment to the product, rather than on design. There was also an acute awareness of the necessity for commitment from stakeholders outside of the S/390 product set. Customers would have to accept the programming model that CB presented, which required support from third party application development tool vendors, and the implementation of CB on Windows NT would have to be successful in order to drive customer exploitation.

**Product Plan**

CB/390 was divided into nine functional areas, each composed of a large number of features and managed as a sub-project with its own team leader. For each area, the CORBA standard was studied and compared to the implementation on other platforms prior to deciding when to port code rather than custom-design it for S/390. Features were the unit of tracking, in order to communicate delivery dates and coordinate dependencies. Delivery was staged into a sequence of builds, as is usual, to facilitate the overlap of process steps. A joint customer study was scheduled, followed by testing and feedback from a second larger beta test group before general availability. The intent was to have the joint study customer in production at general availability.

There was a gross division between base services, required by the CORBA standard in any implementation, and optional services which could be implemented at the discretion of each platform. S/390 also added features within each subset of services intended to differentiate CB from other object environments. Inclusion of services in the base set beyond those defined as base by CORBA required negotiation with the other platforms. For obvious reasons, the base services were prioritized ahead of the optional services that the architects had decided to support. The base services had more of a system-level character, with high cohesion amongst the elements, whereas the optional services more closely resembled independent applications layered atop the base. In addition to the high internal cohesion, all of the base services were considered hard requirements for the initial release, limiting the ability to modify the product in the face of overruns. This ability was also limited by marketing considerations: a single marketing message across all platforms was strongly desired (“deploy anywhere”), although the decoupled schedules of the individual platforms made this difficult to achieve. As the project progressed, most of the optional services initially targeted for the first release were deferred to later releases.

**Process Plan**

The modified waterfall process used by the organization was the basis for CB/390’s process; no variances from the standard process were identified. Thus, the same types of
defect data were collected for CB/390 as for other development projects, and all code was
time reviewed prior to integration. Downstream testing (component test, system test,
performance test) was performed on official (versus internal) builds. Since it was
expected that the normal process would be used, no new process measurements or
ownership arrangements were made.

The implementation of this process varied considerably from the norm, however. The
development and component test teams were required to use a different source code
language (C++) for the majority of their work, store the code in a new library system
based on an unfamiliar operating system, use a different integration process than had
been used for DSOM, use a new database for managing design changes, and use a new
defect tracking technique. Integration was done more frequently (two-week intervals)
than had been used on DSOM, although the extra builds were not released outside of the
development group. One build per month was officially released for testing by other
groups. While there were some experienced people transferred from DSOM, no
investment in formally supported process tooling had been made for DSOM. None was
planned for CB/390 either; CB project and functional management assumed that it was
already in place or could be created informally.

In order to have customer testimonials available at the appropriate time, a joint customer
study was scheduled. An external customer under the relevant legal agreements received
the code as-is at various points as a beta test site, and provided feedback to the
developers. The original project plan was to release the first beta version to the joint
study customer at the very end of the development schedule, after most component
testing had been completed and system test was well underway. This schedule was later
changed to enable product changes to be made in response to the joint study customer’s
feedback, and to obtain feedback from a second group of beta customers prior to general
availability.

![Diagram of CB/390 schedule]

**Figure 18: Early Component Broker/390 Schedule.**

**Organization Plan**

The organizational context for this project was very different from the traditional one
used in OS/390 development, although the organizational structure was only slightly
different from the traditional one.

Since this project was similar in size and scope to those for new operating system
components, the usual approach of creating a new development organization was used.
Typically, this consists of one to five departments for the initial effort, scaling back to a single department for follow-on releases. CB/390 started with two departments, consistent with the recent historical relationship of effort estimates to staffing, that were responsible for the detailed design, coding, unit test, and component test process phases. As noted earlier, architects from the S/390 software platform design group were assigned to shepherd the project requirements and design to speedy realization. The normal downstream test organizations were to be used, with some personnel dedicated early to CB/390 in order to prepare new testing environment infrastructure. All parts of the development organization indirectly reported to the same functional manager, with the exception of the architects.

![Diagram](image)

**Figure 19: Component Broker/390 Functional Organization Overview**

In addition to this functional organization, however, there was an entire cross-functional team (also known as a product development team or PDT) hierarchy in place to handle overall CB product issues. This structure was in place before CB/390 was committed to the product plan, as there were CB implementations on other platforms that started earlier than CB/390. An overall CB PDT handled global issues, and directed the efforts of each platform-specific CB PDT. The CB/390 lead architect had a close working relationship with the overall CB lead architect, and the CB/390 PDT leader was the official CB/390 representative to the CB PDT. The CB/390 PDT leader was an experienced functional manager, who had been the OS/390 release manager during the WLM1 project five years earlier. His influence on CB/390 was wielded through the functional managers in a lightweight project management fashion; none of the functional personnel reported to him, although upper management had made it clear that it was the functional organization’s job to support CB/390.
**Project Evolution**

The initial development plan called for code to be delivered in five stages, spread over nine months, with a further three stages to deliver fixes and respond to any critical needs identified by the joint study customers. Within the code delivery period, additional builds would be created for the internal use of the development group only. After the second stage, schedule and effort estimates would be revised based upon the experience gained during the first four months of development. After the project-planning phase, the development team leaders were in place and functional management had begun staffing the project with more people than expected as such an early stage. Many of these people were unfamiliar with the technical details of the MVS platform, however, so the architects and designers were forced to provide this education internally. Once staff had been assigned, management was convinced by the project leaders that it needed to adopt a hands-off posture and allow the team to work. The architects had been familiarizing themselves with the CORBA standard and the other CB implementations, and were beginning to approach the point of having the knowledge base required to make design decisions. The S/390 platform architecture group (SDC) required design choices in support of the “ities” that raised the product innovation risk from medium to high, with consequent effects on schedule.

One year later, the staging plan had expanded to eight stages spread over fifteen months. The joint customer study was underway, the functional code cutoff milestone was approaching, and actual code size shipped had overrun the initial estimate by several times. Management concluded that there was serious risk that the project would overrun its (revised) release to manufacturing date, and took a more active role in project management. The resulting shuffling of project management, technical reassessment, and team re-definition took six weeks. The CB/390 PDT team leader was replaced with a former functional manager, another functional manager was assigned to manage the project, and several other new team structures were added to prioritize and assign project...
work. Four months later, the VP changed the project to heavyweight project manager leadership due to continuing overruns. At the time of writing, the project was showing signs of improvement, although it had not yet fully recovered.

![Diagram: Revised Component Broker/390 Schedule]

**Figure 21: Revised Component Broker/390 Schedule.**

An incomplete schedule is shown to avoid divulging future plans.

**Discussion**

The jury is still out on this project's ultimate success or failure in the market; clearly, it is not a sterling example of project management. Like WSM, it highlights some of the dangers involved when organizational structure, process, and culture clashes with the project profile. The S/390 software organization is more accustomed to the role of leading the architecture definition than to catching up with it, yet CB/390 was forced to confront an existing architecture that did not stress the traditional S/390 strengths. The organization's traditional strength of leveraging its skill base for new designs became a liability since the skill base had yet to be built. A number of basic changes to the existing architecture were required to allow it to support the level of "ilities" (scalability, etc.) expected on the S/390 platform. These changes could not be recognized until the architects assimilated domain-specific knowledge, and resulted in delays while the changes were negotiated at the CB product level as well as large increases in project scope.

For our purposes, however, the focus must be on the risk aspect rather than project management. There were two major categories of risk, those external to the project and those internal to it. The external risks were dominated by the extra layer of CB product management, and the perceived lack of middle management commitment to the project. Inclusion of services in the base set to differentiate CB also touched off marketing issues; due to the decoupled platform schedules, the same base service set would not be available at the same time across all platforms. The marketing message was complicated by the perceived need to define the "portability limits" of objects, which would vary based on the set of services exploited and their availability dates on various platforms.

The existing CB product architecture required the S/390 architects and lead developers not only to understand the market requirements and how those could be mapped to S/390, but also to understand requirements from other platforms and the existing implementation. In contrast to the existing process, built for projects that utilize sustaining competencies. CB/390 required disruptive competencies. Further, when the architects found areas that required "ilities" changes, often those had to be negotiated with
a group of stakeholders unfamiliar with S/390. Lurking in the environment was the worry that the divisions owning the other platforms might abandon CB as they had DSOM. This resulted in additional time being spent to obtain buy-in from management and organizations such as Marketing who would be crucial in garnering customer mindshare. Marketing of the other CB implementations was important to CB/390, as they were mindful of the OS/2 experience. CB/NT would have to be successful on its own in order to drive demand for CB/390, and third party application tool support would be necessary for both. Taken together, these requirements overloaded the architects' and lead developers' time. Communication with functional management, who was involved in technical decisions, suffered. Both had negative effects on morale; people assigned early saw long hours, basic process problems, schedule deadlines looming, critical technical people too busy doing "other things" (technical gatekeeping and ambassadorial duties), and no end to it in sight due to the apparent lack of middle management engagement. Newcomers quickly learned this from those already on the project.

The major internal risks were technical requirements definition, development skills and process, perceived lack of management commitment, and worries that the short schedule would lead to process shortcuts. The first two were compounded, oddly enough, by aggressive staffing. The others had reinforcing effects on each other.

It was known when development started that the architects had not yet finished their own learning curve, yet they were expected to absorb a large staff of programmers and testers while still educating themselves. Since staffing is traditionally the most difficult hurdle in starting a project in this organization, everyone assumed that some way would be found to utilize the extra staff until "real work" could start. What happened instead is that people attempted to start before the project was ready, and ended up doing massive rework as early decisions were revised. Even such basic issues as packaging were unclear in the beginning. DSOM had used many ad-hoc methods to perform functions such as compile code; while this is fine on a modest-sized porting project, it is often inadequate when scaled up. While the process infrastructure did not exist, and people were staffed on the project that could have worked on this, schedule pressure coupled with functionally oriented thinking meant that they were considered too important to waste on process tooling. In the end a substantial portion of the process tooling used was developed within the group, often by lead developers, once the ad-hoc process was proven to be inadequate to service the larger CB/390 project. The value of development process tooling is de-emphasized in this organizational culture, for historical reasons.

As noted in the "Process Plan" section, there were many tools and processes used on this project unfamiliar to the developers. There was enough expertise within the project to define a basic structure for utilizing some of the tools, but experience was lacking. Many of them were used in other IBM programming laboratories, but again for historical reasons had made little inroad in Poughkeepsie. This created one need for education; the inexperience of much of the staff with OS/390 basics was another. Much of this education ended up being supplied by the lead developers and architects, further stressing what was a serious project bottleneck. The aggressive staffing increased the number of people needing education. The relative inexperience of the staff was attributed to
interproject interference; each functional area’s experts are protected from reassignment in mid-project, and they are over-utilized as well, so any staff initially available often have average or lower experience levels.

A final internal project concern was that the tight schedule would lead the development team to take process shortcuts. An unexpected cultural twist turned this into a major headache, as two diametrically opposed goals were pursued by different factions. While the architects took a view reminiscent of Microsoft’s product introduction process [Cusum1995a, pg. 147, Engst1998a], where the first version is dominated by schedule over features and some bugs are expected, project functional management held to the strong cultural tenet in S/390 that any code shipped must be of high quality and must go through the full testing process prescribed in the waterfall model. The CB/390 PDT, rather than resolving these issues, exacerbated them. Together, this had the effect that the CB/390 functional content stabilized slowly as all of the “ities” were addressed, and the schedule expanded to enable testing the larger product. In the end it was a “right to left” schedule, transition to heavyweight project management, and firm general availability date that was used to force convergence and prioritize content.

**Lessons**

CB/390 attempted to enact major changes in process without infrastructure support, while operating in an unfamiliar organizational context, and on a product with market requirements that were significantly different from those its staff were accustomed to. Not surprisingly, its initial schedule and effort estimates were significantly overrun and its scope increased dramatically. The litany of risks generated synergistically negative feedbacks until a crisis of sufficient magnitude to force heavyweight project management was generated. As with the other projects studied, the heavyweight aspect was informal, although in this case it was directly legitimized by upper management.

- **External communication**: management involvement with the project was low until a flashpoint was reached. Even when the conscious decision is made to “back off and let the engineers work”, some amount of monitoring must remain, and the external gatekeepers must allocate time to keep important stakeholders informed.

- **Aggressive staffing can backfire**: groups without strong pre-existing internal cohesion can only absorb so many new members at once due to communication and education requirements [Brook1975a chapter 2]. When this education further stresses existing personnel bottlenecks in the project, progress will often be slower rather than faster. Staffing a project having an aggressive schedule with inexperienced or average programmers exacerbates the problem. Functionally oriented assumptions that “coders” must code rather than helping with test or process infrastructure, even if the design is too unstable to code from, also intensifies the problem.

- **Allow key people to focus**: the experts must have a basic idea of the right problems to solve and a framework for achieving the solution before new staff can be absorbed. Until this basic level of competency has been acquired, proceeding with the project
risks having it spin off out of control.

- **Tools support**: when process changes are made, some of the existing process infrastructure will no longer work. Commonly this issue is brushed aside in favor of technical issues, for many reasons. It must be realized, however, that the process throughput will limit the capability of the project as a system. No matter how well solved the technical issues may be, if they cannot be turned into product efficiently there will be (at least) two effects. In the short term, schedule delays will result. More insidiously, however, effort will be reallocated from production to solving process problems and their quality consequences. The project plan must address tooling issues when major process changes are made.

- **Cultural issues must be identified and resolved early**: the cultural schism between those with a schedule goal and those with a goal to deliver new function having the same “ities” as a 30-year-old platform was never resolved, although it was articulated. Existing process metrics, which stressed quality and progress along a predictable path, continued to be used by functional management. Even when some of the functional areas had begun to understand and agree to metrics more appropriate to the product, there was a strong tendency to revert to old behaviors at any sign of trouble or disagreement. Since cultural differences are slow to change, they are critical to flush out early, settle, and reinforce. Fortunately, the change to heavyweight project management came at a time when the new metrics had been articulated, so that the proper pressure necessary to change the functional orientation could be brought to bear.

- **Look for synergistic risks**: reinforcing cause and effect loops, a.k.a. vicious circles, are a very serious matter. Once triggered, they can take on a life of their own until undercut by some competing feedback system. Lack of (or the wrong kind of) management focus can lead to armies of inexperienced staff being added to a project, who slow the learning process of the experts, and are immediately put to work although basic requirements have not settled, generating code based on incorrect assumptions that must be reworked later, making the project even farther behind schedule. Product decisions can also be adversely influenced by process limitations: at one point, developers were creating fixes known to be incomplete rather than increase the number of modules affected. This occurred because the immature process for shipping fixes did not scale well.

- **Projects with major changes in more than one area of product, process, and organization need more project planning and project management attention, not less**: organization and process changes make a project especially vulnerable, because of the infrastructure and cultural assumptions that are invalidated. Note that the official organization, embodied in the organization chart, is not the measure of organizational context or change. The reporting and incentive structure, which determines the informal organization, has a greater influence than the formal organization. Further, the organizational context within which the project operates must be considered as well. Often a heavyweight project manager will be required to
maintain the proper project focus. It is easier to implement this early in the project rather than allow it to veer off course and require later rescue.
Applying the Suggested Approach to S/390 Software

The *de facto* assumptions for projects in this organization include the following.

1. The existing waterfall process and functional organization structure will be used. New, large projects might result in the creation of new departments during the low level design, code, unit test, and component test phases of the initial release.
2. The upstream project selection process usually selects projects for which the existing process and organization are sufficient. Projects are competence-enhancing rather than competence-destroying.
3. Projects are incremental upgrades to an existing product.

As mentioned earlier, this organization also has a culture that:
- values risk aversion
  ("take any reasonable risk, as long as you deliver on your commitments")
- values process stability, codification, and the use of the same process across the entire organization with only minor changes from group to group
- is milestone-oriented
- has a strong functional orientation, and a corresponding dependence upon formal specifications for information transfer
- assigns high importance to delivering all committed function, with high quality relative to application-level software and most other platforms
- uses process metrics designed to deliver the "ities"
- normally manages projects with low to moderate product risk

An enduring pattern within this organization is that business needs periodically override the project selection process, allowing projects (usually one per release cycle) into plan whose risk profiles strain the capability of the existing process and organization. These projects become the lightning rod or "problem" project for the release, drawing a disproportionate share of management's attention. [vonH1990a] actually mentions difficulties experienced by computer companies like IBM's S/390 organization: interproject similarities translate over time into an implicit understanding of how changes propagate, and this tacit knowledge reduces the organization's ability to respond appropriately to novel projects.

"These authors suggest that one of the primary sources of the difficulties many firms encounter in their attempts to respond to innovation problems of a novel type is their continued reliance on historically derived but now suboptimal divisions of the problem-solving task.

The way that changes to a given task are likely to propagate across a task network can also often be predicted by project personnel who have experience with similar projects.

...in the instance of routine innovation projects, it is reasonable to expect that project participants can predict which tasks are likely to be the source of
important new information, and which other tasks in the network are likely to be
affected by that new information.

...In the instance of "very novel" projects, an ability to predict the source and
pattern of problem-solving at the outset of that project is equivalent to saying
that we are able to predict the unexpected -- not a very promising prospect.
However, one can still improve task partitioning with respect to problem-solving
interdependence under these conditions by partitioning as the project unfolds."
[vonH]i990a

The fit-based approach is recommended for this set of projects. The projects normally
selected have low to moderate product risk as their predominant risk factor, so by
definition they are a good fit with the existing process and organization structure, and
there is no need for this extra level of analysis.

**Advantages of the Fit-Based Project Architecture Approach**

**Both Project Structure and Change Timing Issues are Addressed**

Both questions (how to structure projects, and how to recognize the need for
restructuring), as well as the derived question (when is the best time to change the
structure) have been addressed. Structure, including all of the project's architectural
elements, is chosen to fit the project as well as possible within the organization's
capabilities. Criteria for recognizing that purchasing information is appropriate have
been presented. Re-evaluation of fit as the project unfolds and the timing of structural
changes is based on pacing events, either external or internal to the project, and the
midpoint transitions of major project phases delimited by the pacing events.

**Low Implementation Cost**

A version of the fit-based approach, tailored for S/390 software, would require only a
determination of the current process and organization strengths (management of
moderate/low-low-low project risk profiles, with product risk highest), and details
regarding the "ideal" process and organization (since these are de-emphasized by the
S/390 software culture). No specialized tools are required for its implementation, and
since it would be applied only to "very novel" innovation projects the effort required to
perform the analysis should be well within the capacity of the organization. The largest
cost is likely to be training in organizational and process diagnosis, since these skills are
generally lacking or unpracticed in S/390.

**Project-Wide Scope is Encouraged**

For an organization such as the one under study, with a history of process codification
and strong functional culture, the fit-based approach has the advantage of encouraging
cross-functional scope when architecting projects. This helps to remove some of the
perceptual and cognitive barriers that result from the strong functional culture, and it
more closely matches the top-down viewpoint of management concerned with a project's
"health". Taking an example from CB/390, some risk survey respondents talked about
the lack of build tools, others noted that new test infrastructure was required. These were
collapsed into the process risk category in the case study. Of the twenty-two individual risk items identified in that survey: four were completely external to the group, three did not fit in well anywhere (the infamous "miscellaneous"), and the other fifteen collapsed very nicely into the four risk categories.

**Adaptation is Encouraged**

S/390 has begun to recognize that the overhead of the normal process is too great for small projects, for example. One of the key advantages of both the risk-based and fit-based methodologies is that they encourage adaptation. "Using the risk-driven approach, one can see that the answer is not the same for all projects and that the appropriate level of effort is determined by the level of risk incurred by not doing enough" [Boehm1988a].

**Process, Organizational, and Environmental Aspects are Highlighted**

"Very novel" [vonHi1990a] projects represent a special challenge to S/390, because it has considerable historical experience in managing its normal projects. Process and organizational infrastructure, normally taken for granted as being in place, tend not to be thoroughly or objectively examined. Organizational factors such as those documented in [Ancon1992a, Ancon1992b, Hackm1990a, Thamh1987a], to the extent that they transfer to S/390, would be useful additions to management’s arsenal of knowledge. The idea of fit between task uncertainty and processes, as well as the effect of decisions made at macro-organizational levels, would also be useful additions [Drazi1985a].

**Retrospective: S/390 Case Comparisons by Project and by Sub-project**

The following figure illustrates several different points from the cases:

1. The S/390 software organization tends to partition based on product functionality. Product risks are usually not changed by decomposition alone; rather, decomposition exposes the divergent risk levels in each sub-project, enabling more effective project management. WSM, WLM1, and WLM4 were decomposed to separate the management of high-risk algorithms from other parts of the project. Where the functions cleaved cleanly along the functional plane, decomposing in this way focused attention where it was needed. The functions that did not cleave well along this plane were harder to manage and integrate as a result.

2. Organizational and environment risk is tolerated relatively well. This organization makes extensive use of informal gatekeeping and ambassadorial roles to manage these risks. The high degree of job specialization, which requires downstream integration, also requires the existence of formal coordination mechanisms. In situations where these risks are medium to high, there would probably be benefit in formalizing these roles as was done in the WLM1 and WLM4 cases. In WLM1, senior personnel were assigned to manage the relationships with each product (CICS, IMS, RMF, ISV performance monitors), each functional area (component test, system test, performance test), and with customers participating in the design of externals. In WLM4, the JES2, SDSF, and DNS liaisons were informal, while other product interfaces (RMF, ISVs) and externals design relationships were formalized.
3. If a project is decomposed, this must be a formal, legitimized decision. The WSM work stoppage was a result of disagreement within the project about whether or not to continue using the ad hoc partitioning that had developed, not due to intraproject organizational fit problems. The lack of external fit did exacerbate the situation, however. The WSM experience, coupled with personnel continuity, led to an explicit decision on project structure for both WLM1 and WLM4. CB/390, like WSM, suffered from friction between the differing structures of the formal and informal project organizations. Lack of a single strong project leader further compounded this situation, and ultimately a strong project leader was needed in order to align the organizations.

4. The projects studied adapted in a way that increased internal fit, usually at the expense of external fit. The job of maintaining the relationship between the project and the encompassing organization when external fit is lowered falls by default to the project leader. In WSM and CB/390, where this issue was no sufficiently addressed, lack of external fit blossomed into a set of problems with project-wide scope. In WLM and WLM4, where the external fit problem was actively managed, these problems were avoided.

5. The projects studied adapted in a way that increased internal fit between product and process, allowing organizational fit to suffer if necessary. From an organizational standpoint alone, using iteration to develop the WSM algorithms created stresses due to role ambiguity in the organization. Yet iteration was used anyway, in order to manage product risk more effectively via process adaptation. Although the internal organizational stresses generated a crisis, iteration survived as a process adaptation in the WLM group. In contrast, CB/390’s process customizations were ignored by the formal product delivery process even after it became apparent that project throughput was suffering. The installation of a heavyweight project manager was required to begin adaptation of the formal process to CB/390’s specific needs.

6. Successful projects partitioned and adapted their work to concentrate risk in areas where mechanisms existed to mitigate it: the product and environmental categories. They actively moved away from process and organizational risk. The product and environmental risk categories are those with which the larger organization has experience, through project management, gatekeeping, and ambassadors. For example, separating the externals from the algorithms in WLM1 allowed the assignment of focal points for inter-product problems.
<table>
<thead>
<tr>
<th></th>
<th>Process</th>
<th>Informal Org</th>
<th>Risk</th>
<th>Fit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal</td>
<td>Iterative</td>
<td>Functional</td>
<td>Project-oriented</td>
</tr>
<tr>
<td>WSM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td>●</td>
<td>●</td>
<td>M</td>
<td>H</td>
</tr>
<tr>
<td>Decomposed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Externals</td>
<td>●</td>
<td>●</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Algorithms</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decomposed and</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adapted</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Externals</td>
<td>●</td>
<td>●</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Algorithms</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WLM1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td>●</td>
<td>●</td>
<td>M</td>
<td>H</td>
</tr>
<tr>
<td>Decomposed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Externals</td>
<td>●</td>
<td>●</td>
<td>M</td>
<td>L</td>
</tr>
<tr>
<td>Algorithms</td>
<td>●</td>
<td>●</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>Application</td>
<td>●</td>
<td>●</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>Decomposed and</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adapted</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Externals</td>
<td>●</td>
<td>●</td>
<td>M</td>
<td>L</td>
</tr>
<tr>
<td>Algorithms</td>
<td>●</td>
<td>●</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>Application</td>
<td>●</td>
<td>●</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>WLM4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td>●</td>
<td>●</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>Decomposed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-batch</td>
<td>●</td>
<td>●</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Resource affinity</td>
<td>●</td>
<td>●</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Initiator mgmt</td>
<td>●</td>
<td>●</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>Decomposed and</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adapted</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-batch</td>
<td>●</td>
<td>●</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Resource affinity</td>
<td>●</td>
<td>●</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Initiator mgmt</td>
<td>●</td>
<td>●</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>CB/390</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td>●</td>
<td>●</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>Overall</td>
<td>●</td>
<td>●</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

Figure 22: Summary of Case Project Decompositions, Adaptations, Risks, and Fit

No decomposition is shown for CB/390, although its development group (low level design through component test) was divided along internal product functional lines into nine working groups. These groups did not cleave cleanly along any lines other than product decomposition, so they functioned primarily as working groups to manage the product complexity rather than planes along which to customize aspects of the project architecture.
Conclusions

“Patient: Doctor, it hurts when I do this!”
“Doctor: Then don’t do that.”

The casual reader might advocate simple avoidance: preventing selection of these projects in the first place, so that only projects which fit well with the preferred process and organization are injected into the system. The problem is, of course, that shifting strategic directions will inevitably result in “non-traditional” projects being undertaken. The question is, how will the organization cope with them?

For the S/390 software organization, the preferred mode has been to use its waterfall process and functional organization for all projects; the existing integration and coordination mechanisms have been expected to resolve any problems. When things have remained in this state (WSM, CB/390), the projects have experienced major problems. The process-phase milestone orientation of project monitoring delays the realization by product and project management that problems exist, by preserving the illusion of progress until a milestone is formally missed. This contrasts with an iterative approach:

“The older concept of ‘write the spec, write the code, test the code, maintain the code’ as a life cycle idea is very misleading as to how near completion you are. You see lots of projects that stay 80 to 100 percent complete for 80 percent of the time. So we’ve developed this idea of a … milestone process, where you try to pick out some of the more difficult things to do, you finish them 100 percent, and then you reevaluate where you are… You do all the … things that, if that set of features was the product, you’d be ready to ship it… You know much earlier in the process when you’re in really bad shape… You know where you are.” [Cusum1995a, p. 277]

Once projects have adapted their elements, sacrificing external fit in favor of internal fit (WSM, WLM1, WLM4), things have progressed more smoothly. In essence, a small section of the functional organization cordoned itself off, temporarily adapted to project requirements, and re-integrated into the larger organization at project completion. Some adaptations, such as algorithms verification test, were legitimized over time and retained as a local process customization, becoming the preferred mode for WLM algorithm modifications. Others, such as iteration, are not needed frequently enough for the larger organization to adopt or even maintain competence in them. Adaptation is not a panacea, however. No amount of adaptation by these non-traditional projects will make them progress exactly like a traditional project, despite the wishes of release managers. Adaptation can reduce risk, but it cannot transform a project from non-traditional into traditional.

Risk reduction through conscious choice of project architecture provides the overall organization with a new choice: rather than hoping the existing process and organization can tolerate non-traditional projects, those projects can be structured based primarily on project needs (internal fit) from the beginning. The external fit problems created can then be proactively managed using formal ambassadorial roles. This allows the organization...
to take advantage of process and organizational models whose strengths and weaknesses better match the needs of the project. For sub-projects where the waterfall model and functional organization are appropriate (low to moderate product risk, low process risk, generation of knowledge new to the organization is not critical for project success), the organization's preferred modes can be used. For sub-projects where iterative delivery is appropriate (high requirements and/or innovation risk, significant process risk, new knowledge required for success), that choice can be made – and legitimized – consciously. For cases where decomposition does not partition risk (e.g. CB/390 process problems), that can be recognized and addressed as part of the project delivery plan. Finally, this analysis is can be performed on in-progress projects or during initial project planning.

Recognizing projects likely to stress the existing project implementation style has not historically been a problem within S/390 software; managing those projects has proven somewhat more difficult. The cases studies and analysis suggest the following:

1. **Start with a heavyweight project manager**, to minimize delays caused by the built-in conflicts between aspects of the project architecture. For large projects, or those believed to be particularly risky, this should be a formal position with at least one member from each functional area formally reporting to the team. Note: in every case studied, some form of heavyweight project management was eventually used to bring the project back under control; the more distressed the project, the more formal this position became and the stronger its legitimization by upper management needed to be. It seems logical to start with this organization to avoid or minimize problems from the beginning, rather than requiring large-scale rescue later.

2. **Divide the project into sub-projects based on risk profile**, to provide a natural point of divergence for process and organization to match each portion of the project. **If risks cannot be partitioned by decomposing the project, buy more information.** This might take the form of a limited-scale prototype, or working on process infrastructure while product issues are resolved.

3. **Manage external communication**, so the project does not become isolated from its stakeholders. Determine which external boundaries are important, who can span them (gatekeeper and ambassador skills), and who should span them (personality etc). Allow them sufficient time to do so, by making boundary management an assigned function.

Understand and manage the different types of external communication differently. When external stakeholders must be influenced to provide resources, this requires a high time and effort investment to negotiate and maintain commitment from the other parties. External communication for information gathering is a far more discretionary investment.
4. **Understand the risk management philosophy of processes used, and choose processes appropriate for the sub-projects’ risk profiles.** While many in this organization believe that the waterfall process is low-risk, this is untrue. A waterfall process merely shifts existing risk downstream to the integration point. Since it encourages the expenditure of significant effort before large-scale integration, product-level feedback occurs late in the project, often too late for the current product generation to incorporate new knowledge. When requirements and innovation risks are low, this is usually adequate.

Iteration, on the other hand, is geared toward the generation and incorporation of new knowledge as the product progresses precisely because it encourages integration and testing before the bulk of development effort has been expended. This is why iteration is appropriate in cases of high requirements and/or innovation risk. When iteration is part of the project processes, review the lessons from the WSM case. Iteration requires a substantial departure from the organization’s normal processes.

5. **Monitor the project based on metrics matched to project goals.** Especially when process changes are made, existing metrics may be inappropriate. Consider what each metric actually measures, and whether it is appropriate for each process. New processes often require new metrics.
Bibliography


