CHAPTER 2

STUDY ON PROJECT MANAGEMENT

2.1 Introduction

Effective project management is crucial for the success of any project. Many projects are ending up as failures not because of lack of competent technical professionals or resources but due to the use of faulty project management practices. Therefore, it is imperative to know the latest techniques in the field of project management. So different types of project management techniques are presented here. The study of types of management techniques reveals that concurrent engineering is widely advocated as essential to successful product development programs. Hence benefits of concurrent engineering and its various types are discussed here. Today software has become one of the largest and fastest growing industry. So a methodology for applying concurrent engineering principles to software engineering is discussed in this chapter.

2.2 Project Management

Project Management is the discipline of organizing and managing resources such as people, time and money in such a way that the project is completed within defined scope, quality, time and cost constraints. The management of the resources is often very difficult and requires varying technical skills and philosophy, hence requiring the development of project management. Following are the various typology of new project management [45]:
**Traditional project management:** This is the most common project type. Top-Down planning is used more often and work-breakdown structure is designed before the project is started.

**Co-located program:** In this case, there are multiple concurrent projects, all of them operationally co-located in a single geographical place. Turner and Speiser, 1992 [53] elaborate on this situation, defining program management as the process of coordinating the management, support and setting of priorities on individual projects, to deliver additional benefits and to meet changing business needs. Some of these projects can be interdependent, because these deliver related objectives and share common deliverables, information, resources and technology.

**Multiple co-located programs:** In this case, the difficulty is aggravated by the fact that sets of projects (or programs) are now in geographically distant sites. The distance may create communication-problems among the project managers and also among program managers (if more than one). Face-to-face communication may not be possible. In addition, sharing of resources requires more careful scheduling, since in emergencies a resource cannot be requested and moved seamlessly from one project to the next if they are in different distant sites.

**Multiple traditional projects:** It can also be thought of as a subset of the above, with the exception that there is only one project in each site. Most of the traditional project management considerations apply. The key issue is scheduling. However, there is a potential need for sharing resources to achieve true global optimization.
**Distributed projects:** In this case a single project encompasses several sites. This situation occurs for several reasons, among them scarcity or complementarities of resources, convenience, cost, monitoring capacities, and quality.

As this software development effort exemplifies, a distributed project has many benefits. On the other hand, there are also many problems. The principal issue is the heightened need for communication and coordination of the separate pieces of the same project being developed in different areas. The extra coordination is necessary to schedule the different activities over several sites and concurrently trying to allocate resources. In principle, the only person who has a bird’s eye view of the overall situation is the project manager. There is no overlap of people across sites. Meetings across sites and different parts of the project are, most of the times, accomplished by project managers who are in charge of more than one site. Team members also communicate to clear doubts or create a better relationship with other off-site project members. Interdependence across sites is considerable due to their shared needs for the same resources.

**Multiple distributed projects: discrete locations:** This is the case where many distributed projects as described above are managed by the same project manager. Key issues are very similar to those in distributed projects, with the exception that now the boundary spanning efforts become more inter-site in nature than previously. There is no assumption made about the interdependence between the projects, and therefore potential synergies are not necessarily explicitly recognized or managed. Inter-site resource allocation and sharing has similar problems to other program forms such as multiple co-located programs and multiple traditional projects.
**Multiple distributed projects: shared locations:** This is the general case of the previous category. The fact that some locations are shared simplifies the boundary spanning efforts of individuals who may be participating in more than one project concurrently, helping them to recognize similarities and synergies across projects. Moreover, resource allocation/sharing is some-what simplified for the same reason. Possible advantages could include lesser cost due to avoidance of effort or resource duplication. On the other hand, one should be aware that politically it may be more difficult to negotiate and schedule resource allocation.

In most of the above technique, the activities occurs simultaneously which leads to birth of concurrent engineering.

### 2.3 Concurrent Engineering

Concurrent Engineering [CE] is "a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. This approach is intended to cause the developers, from the outset, to consider all elements of the product life-cycle from conception through disposal, including quality, cost, schedule, and user requirements" [55].

\[ CE \Leftrightarrow \text{[Paralleling of (life-cycle functions)]} \]

Some experts recognize influencing agents of CE as forces of change [11]. Prasad [41] has chosen to divide forces that influence the domain of CE into seven agents (called here as 7Ts): talents, tasks, teams, techniques, technology, time and tools. One of the primary team issues is the decomposition of tasks. The people's issue is the composition of teams. Teams are often used to cooperatively solve the problem.
Technology issues arise due to drive for competitiveness. Examples of popular technologies in CE are soft prototyping, visualization, product data management, design for multimedia, electronic data interchange (EDI), etc. Tools include software, hardware, and networks that make CE practical in today's world of multinational corporations, multipartner projects, and virtual corporations. From the time point of view, CE is an initiative of the product development community that has the goal of reducing the length of the product design and manufacturing cycle time by allowing teams of engineers to develop design modules concurrently from their perspectives [40]. Training also plays an important role in CE. A popular word in the business press is reengineering, meaning, in short, revamping the processes by which one satisfies customers needs. From a business angle, CE means reengineering the product development process so that tasks are organized concurrently [5].

CE is founded on eight basic principles - Early problem discovery, early decision making, work structuring, teamwork affinity, knowledge leveraging, common understanding, ownership, and constancy of purpose.

2.3.1 Benefits of Concurrent Engineering

The following [5] are the benefits of concurrent engineering:

Rationalization: The CE approach ensures that design will be rationalized for the capabilities of the manufacturing system, which is going to convert the intent design to reality-a physical part that can be produced through a physical process. The team focuses on a common goal instead of conflicting departmental goals.

Working Parallel: The primary CE advantage is that many processes occur
concurrently rather than sequentially-things are done simultaneously rather than in a linear fashion. For example, manufacturing engineers can begin to evaluate tooling while the design is still being finalized. Similarly, the documentation process can start virtually on the same day as conceptual design.

Reducing lost time caused by communication breakdowns: Since the multifunctional teams have the same objectives, the teams strive to be more open and available to input each other. Communications tend to be more frequent and less formalized. In a linear process the team is divided. A lot of time is spent on needless conflicts.

Preempting errors and spotting problems early: With concurrent engineering, there is a much greater chance of preempting errors and spotting problems. When products are finally released for production, there are fewer changes, while the ones that do crop up are usually minor.

Flexibility to accommodate changes: When work is done in parallel, it provides the team with flexibility to accommodate changes and the ability to adapt quickly to changed situations or requirements. This ability to accommodate changes at any stage in the product cycle (beginning, middle, or end) provides a smooth flow of information among the involved organizations.

Providing the best overall input: The benefits of concurrent engineering reach for beyond the usual collaboration between engineering and manufacturing. CE brings concentrated resources to bear on the task. For instance, various individual disciplines also benefit from working in parallel on product development. In a
growing variety and complexity of the products that are manufactured today, close cooperation between mechanical and electronic design is critical to speedy development. Cooperation is fruitful to other disciplines as well—industrial design, conceptual design, structural analysis, technical documentation, etc. When teams and disciplines cooperate with each other, the mutual outcome is likely to be better and faster.

Decreased occurrence of obsolescence: The large interval between system development and deployment during traditional product design increases the possibility of obsolescence before the product is ready to be marketed. In CE, since time-to-market is reduced, the effect of obsolescence on the life-cycle cost is greatly minimized. As a result, CE not only saves development cost due to shrinking time and schedules, but it also reduces non-recurring costs (such as parts qualifications, tooling, inventory, stocks, manuals, support equipments, etc) that would have incurred otherwise.

Cross-training: a multifunctional group can help each other get up to speed quickly. Cross-training reduces the learning curve.

Better use of scarce technical resources: If the product development cycle can be shortened, then project personnel can be made available for the next project sooner, thus improving the company’s use of scarce technical resources.

Other Benefits: In addition, several of the firms that have recently adopted the practice of concurrent engineering reported reduced lead time for creating bid
proposals, shorter product development times (in design phases): reduced product development costs, parts reduction, lower inventories, fewer rework orders, smaller scrapples, improved quality, and reduction in production costs [McKnight and Jackson 1989] (during fabrication, manufacturing, and assembly).

CE emphasis the need of managing both activity and information concurrency. Activity concurrency refers to the tasks and design activities that are performed simultaneously by different people or groups. Information concurrency refers to the integrated, or team, approach to development in which all the concerns of the different functions—the customer, R & D, design, engineering, manufacturing, sales, and service—are addressed through a flow of shared information.

2.3.2 Activity Concurrency

Within-stage overlap of tasks is the simple form of activity concurrency fig 2.1. As it is commonly practiced within a stage such as detailed design, the problem of

![Diagram of Detailed Design with Modules 1, 2, and 3]

Fig 2.1 Within Stage Overlap [23]
designing, say, a new model TV is subdivided into modules (e.g., power supply, tuner, projection system) that are designed in parallel. Although less common, parallel activity can also be practiced within other stages, such as the requirements stage, where different functions—marketing, R & D, and software engineers—work in parallel to specify customer requirements.

Across stage overlap involves concurrent activity across different stages of the development process, such as high-level design, detailed design and testing fig 2.2.

![Diagram of development process](image)

**Fig 2.2 Overlapping Across Stages [23]**

For example, some firms overlap detailed design and testing activity. Instead of waiting until all the design modules are completed before testing, they overlap and time-compress this activity by simulating the presence of other components and incorporating the testing into the detailed design phase. Few firms attempt this activity because, although potential integration problems are exposed earlier, the risk
and cost of failure are high. Hardware/software overlap occurs when software must be imbedded into a larger system, such as a TV set, which integrates hardware and software fig 2.3. Once the high-level design and product specifications have been determined, design of hardware components- exterior box, the projection tube and circuit boards-can be performed in parallel with the software required to drive the system and its features.

![Hardware Design](image)

![Software Design](image)

**Fig 2.3 Hardware/software overlap [23]**

Across project overlap presents a different type of concurrency challenge for management fig 2.4: components designed for one product can be reused in current and future product releases. That is, the design activity for two or more products takes place simultaneously through the design of reusable, or multiuse, components. For example, in the design of software to support a new feature for TV, such as multichannel screen displays (or picture-in-picture), a robust software module that can deliver the feature in other models without reprogramming is a productivity enhancing form of concurrency. Activity concurrency, in all its forms, is supported in varying degrees by information concurrency.
2.3.3 Information Concurrency

Information flows to support concurrency take three forms, which we call 1) front loading, 2) flying start, and 3) two-way high bandwidth flow. Front loading is the early involvement in upstream design activities of downstream functions or issues-process engineering, manufacturing, customer service concerns, and even future market developments. Front loading provides an early warning about issues that, not considered, could lead to costly redesign and rework later. In hardware, for example, front loading of manufacturability problems can keep them from becoming major show stoppers when the design is transferred to production. Front loading of information about the direction of developing technology can add the creation of more robust and reusable designs.

Flying start is preliminary information transfer flowing from upstream design activities to team members primarily concerned with downstream activities. For example, partial design information about new materials to be used in a product can
provide a jump start for process engineering and help compress the time in process
design activities.

Two-way high bandwidth information exchange is intensive and high communication
among teams while performing concurrent activities. The information flow includes
communication about potential design solutions and about design changes to avoid
infeasibilities and interface problems. Two-way high bandwidth flows differ from
front loading and flying start, which tend to be unidirectional flows of information
from one function to another. For example, the hardware/software overlap diagram
displays a situation in which two way high bandwidth flows are critical: teams
involved in concurrent hardware and software design need to have a steady flow of
information among the groups to prevent potential integration problems [23]. The
definition of CE in terms of activities and information provides the framework around
which the concept of Concurrent Software Engineering (CSE) is built [23].

2.4 Concurrent Software Engineering:

Software has been developed for commercial purposes since the late 1950's. Initially,
writing software programs for computers was viewed as the creative act of highly
skilled artisans working in isolation. Some in the industry considered software
design is an art and dismiss software engineering as an oxymoron [23].

As the industry grew, however, the projects became too large and complex to be
carried out by an individual; budgets and time constraints were exceeded with
alarming regularity. Efforts to manage software more effectively led to a discipline of
software engineering for more rigorous project management in which division of
labor and structured problem-solving techniques support creativity. Software engineering sought to eliminate bad practices and to impose formal methods for rigor in specification and design of software. In addition, it sought to enhance productivity and quality by providing computer aided software engineering (CASE) tools to developers for all stages of the process [23].

During 1993-1994 Joseph D Blackburn and et al., 1992 assessed the state of software development by surveying managers of over 150 projects in the United States, Europe, and Japan, and by following up with field interviews [23, 6]. They asked them to review a recently completed software project and reflect upon the tools and techniques that were most useful in reducing project completion time. Most of these projects are large, ranging in size from 2000 to 1 million lines of code. By their own assessment, software managers perceive their organizations as becoming faster and more productive. When asked to estimate the duration of the project had it been undertaken five years earlier, the managers indicated that, on average, the project would have taken 30% longer [23].

To learn the degree to which CE principles were important, the software managers are asked to rank the following factors with respect to their effectiveness in reducing overall software development time: the use of prototyping, better initial customer specifications, use of CASE tools, concurrent development of stages or modules, less rework or recoding, improved project team management, better testing strategies, reuse of code or modules, changes in module size and/or linkages, improved team communication, and better programmers or software engineers [23].
There is clearly some evidence that concurrency practices are having an effect on reducing development time. In the American and Japanese firms in the survey, code reuse—a form of concurrency across projects rather than within—is rated the most important factor. Concurrent development is the second most important factor in Japan, and better customer specifications, an information support activity for concurrency, was highly valued in the United States [23].

Within-Stage Overlap

Parallel design activity is a widespread practice in the detailed design stage of both hardware and software; modularizing the problem into elementary building blocks to be solved by an individual or small team. Brooks [8] observed that increasing the number of modules has adverse side effects: communication flows between teams working in parallel increase as a function of the square of the number of teams. This is confirmed in the field interviews with software developers who stated that as parallel activity in coding increases, the probability of interface errors and integration problems also increases, expanding the overall project completion time [6].

Increasing the level of within-stage concurrent design activity clearly increases the risk of integration and test failures, creating additional information management challenges. The goal of architectural modularity is to minimize the need for communication among modules, but to the extent that this is not achieved and communications needs rise, management's priorities shift to practices such as synchronicity. The need for synchronicity to control communication flows increases in direct proportion to the lack of success in module definition, or architectural
modularity. For synchronicity, the key information support activity then becomes two-way high bandwidth information flows [23].

Across Project Overlap

Software rarely exists in isolation. Like hardware, software is updated, redesigned, and elements are reused in new products. Projects have genealogical links to predecessors. Design reuse is a way to achieve concurrent design activity across projects whose development cycles do not overlap. When a component is reused in a subsequent product, the original design work for a component is a form of virtual concurrency because in the initial effort, one is simultaneously doing the design work for all future products in which that component is used. Reuse is difficult to achieve because it requires stability, whereas product development is an environment of constant change and this tends to scuttle most reuse programs.

Most software reuse strategies that have been described are reactive; reuse occurs more by accident and imposed time constraints than as a planned process. Planned, proactive reuse is a management responsibility; providing a clear vision of requirements for future products that sustains stable objects and platforms. To foster reuse, the key design practice is the same as for within-stage concurrency-architectural modularity—but for a somewhat different reason. Across projects, architectural modularity is needed to provide stable, well-defined objects that are robust enough to transfer across platforms and to withstand changes in other objects (within stages, architectural modularity is needed for independence of objects so work can go on simultaneously with little intervention).
The key information flow necessary for achieving architectural modularity is front loading. Across projects, front-loaded information benefits designers by providing good forecasts of the form and requirements for future modules. This information flow helps them develop more robust and reusable designs.

To achieve the planned reuse, many practitioners expect O-O programming to be the silver bullet. Hardware reuse is based on stable, well defined components and analog in software is an object with precisely defined interfaces and functionality. However better tools may provide only a partial solution [23].

Across-stage Overlap

The concurrency is within an individual project and explore the overlap of activities across development stages. Few software firms attempt this type of across-stage concurrency. Most of the firms follow some version of the sequential waterfall model [6]. Beginning high-level design activities before requirements definition has stabilized, increases the risk that changing specifications will require redesign, and the cost of reworking a stage can be exorbitant. The manager at a large German electronics firm stated emphatically that stage overlap was incompatible with software development due to the instability of early stages. Eppinger [50] notes:

... stage overlap succeeds in reducing overall task time only when earlier iteration (overlapping) eliminates later iteration (feedback) which would have taken longer.

Examples from hardware development reported by Clark and Fujimoto [12] show a strong correlation between the degree of stage overlap and shorter development
cycles. Some software developers take the risk, reasoning that letting modules progress to the next stage is too advantageous in terms of time gained to forego. Synchronicity is the key activity that supports concurrence across stages. Synchronicity is critical because when coordinating activity across stages, interdependencies cannot be eliminated, they must be managed. To facilitate synchronicity, all three types of information concurrency are important. Two-way high bandwidth flows are needed to keep the process from getting ‘out-of-sync’ and to compress the time between occurrence and detection of problems. A flow forward of information downstream or flying start, is needed to begin work on downstream stages. Front loading is critical in the requirements definition and high-level design stages to firm up product requirement; user requirements tend to be volatile and changes in specifications are one of the chief causes of project delay and rework.

Microsoft is a case in point of how software firms can employ across stage overlap. Cusumano [14] reports that, in an effort to reduce the time-to-market for large software applications, Microsoft actively promotes management practices to overlap across stages. Having abandoned the sequential waterfall model, they have adopted as management practice a procedure to “synchronize and stabilize.” Specifications, development, and testing are all carried out in parallel but they synchronize with daily builds. Some software organizations use tools to overlap activities in the early stages of the process. One popular technique for front loading of requirements is rapid prototyping, which presents a potential user with a realistic interface whose functionality is simulated [23].
Hardware/Software overlap

Increasingly, software is embedded into a larger system which integrates hardware and software. In an integrated system of software and hardware, the complex interdependencies create coordination problems that can cause the lead time to exceed that required for the independent design of subsystems.

Delaying the integration of hardware and software until the first testable hardware prototype reduces system quality and extends the time. As Morris and Fornell [36] note: “Bringing hardware and software together at this late state is troublesome for several reasons. Engineers have little time to correct design problems, and fixes are more costly than they are earlier in the design process. Options for revisions are much more limited; because of the rigidity of the hardware design changes are usually made in the software, at the expense of system performance”.

To achieve CSE, active project management is needed to coordinate hardware and software design efforts—that is, synchronicity. As with across stage concurrency, all three forms of information concurrency are required to support an interactive parallel process. Front loading is critical for up-front involvement of software concerns in hardware design; two-way high bandwidth communication can provide insight into the origins of changing requirements so that developers can learn to predict changes.

Because of the complex software/hardware interfaces, changes in either one usually translate into changes in the other. Long, repeated rework cycles are the chief cause of cost and time overruns. Although these problems certainly occur in software (or hardware) designed alone, the interdependencies in firmware increase both the
frequency and severity of problems. As is the case with stage overlap, two-way high bandwidth information transfer can provide early detection of interface problems. This may not, in itself, decrease the frequency of quality problems, but it can reduce their severity by locating them closer to the source and preventing the 'spread of infection' to other parts of the design. Since instability of requirements specifications is a major problem, CSE can address this with simultaneous development of hardware and software requirements [23].

2.5. Discussions

The project development methods discussed here show the currently following methods by the product development companies. Concurrent engineering represents one of the most significant contemporary trends in new product development. It is a way of attaining three business objectives simultaneously: reducing time to market, improving product quality and lowering development costs. The benefits and problems of concurrent engineering discussed here shows some more methodologies and tools are required for effective use of concurrent engineering. The method of concurrent software engineering discussed here shows concurrent engineering principles can be used for software development.