Globally distributed product development using a new project management framework

Partha Pratim Ghosh *, Jacob Chandy Varghese

Lucent Technologies (India) Ltd, Golf View Campus, Wind Tunnel Road, Bangalore 560017, India

Received 23 October 2002; received in revised form 7 February 2003; accepted 11 May 2004

Abstract

There is a growing pressure on corporations to streamline operations involving multi-site development. In this article, we propose a new project management framework that can be used for managing and tracking distributed development of a large product efficiently. We assume that the development team is organized as Centers of Excellence (CoEs), which may be geographically distributed. The framework described in this article, is capable of presenting diverse views (e.g., Feature, Load, and Release) of a product to its stakeholders seamlessly. It also streamlines communication between the CoEs. A product plan, designed using this framework, offers high resilience to requirements changes during the development cycle. The planning issues for a large (over 200 features) product are described in detail next in the context of the framework. We conclude with a brief scope for future work. The paper chooses examples from the Software domain though the approach is general enough, and is applicable to other disciplines as well.

© 2004 Elsevier Ltd and IPMA. All rights reserved.

Keywords: Product management; Framework; Center of excellence; Release; Loads; Features; Load contents; Requirement volatility; MS project; Circular dependency

1. Introduction

With technology leapfrogging and ushering its benefits bounteously into society, the expectation of its beneficiaries from the technology-oriented organizations is increasing more than ever before. This is forcing corporations to streamline their development strategies towards achieving faster turn-around time for product development at a much lower cost to sustain their competitive advantage. With globalization at the backdrop, capitalizing on the geographically distributed skills to achieve high quality product deliverables in a much shorter span of time is becoming a necessity today.

Literatures abound in the area of distributed project management including tools that facilitate planning. PERT/CPM approaches have been in existence for quite sometime but they are limited in terms of their analytical power as cited by Cho [1]. Cho [1] had primarily concentrated on aspects such as Process Restructuring and Process Modeling of Distributed Project Management. Process Restructuring typically deals with the structure of the information flow in a complex project while Process Modeling captures the behavioural aspects of processes over timelines. Other efforts towards Process restructuring include approaches proposed by Steward [5] such as Design Structure Matrix or DSM that models information flow across tasks based on large-scale system decomposition.

There have been focused activities towards development of project management tools. Rojers [3] developed a computer aided design tool to support assisting in large-scale system decomposition. As cited in Stallings [15], some researchers have also come forward with an “Intelligent Agent” based project management framework. This is an integrated project management framework where plan and design may change while the plan is in execution. Notification about any change is passed as ‘messages’ to software agents, thus, minimizing human intervention. According to Petrie et al. [2], the

*Corresponding author. Tel.: +91-80-2550-4834.
E-mail address: ghosh_p@hotmail.com (P.P. Ghosh).

0263-7863/© 2004 Elsevier Ltd and IPMA. All rights reserved.
doi:10.1016/j.ijproman.2004.05.008
problem of using traditional tools such as Mac Project, or MS Project, is that they are inherently based on single user model of planning, with very primitive change notification mechanisms. Centralized decision-making vs. distributed decision-making is what is highlighted in the first part of Petrie et al. [2].

One of the most important challenges in product development in a distributed environment is to ensure proper co-ordination and control [6]. Many recent studies [7,8] emphasize the use of Information Technology in tackling project management issues in a distributed environment. Others [9,10] explored the use of diverse Communications Technologies to address the co-ordination issues in a distributed environment. Less technical in nature, Lori Anschuetz [11] elaborates on how sharing of corporate culture, gaining insights into clients, building trust and bringing in professional synergy can enable efficient co-operation and effective results for ‘virtual’ organizations. Much of the work in the area of distributed Project management lacks the rigor in dealing with the concurrency issues [12–14] when the same project involves development across multiple locations.

We propose a simple approach towards distributed project management in this paper that is broadly based on Process Restructuring. We believe that, by establishing a suitable framework that helps in automating and consolidating Project Control related information, the distributed/co-located teams are better informed to take planning decisions, thus, giving them more bandwidth to focus on other issues relating to a new product. We define a protocol as well as an interface for communication between development teams for a product. By virtue of this, the existing skills and framework that an organization may have developed over the years can be readily used.

2. Organization structure and product development

One of the common ways of structuring a large product organization is to create Centers of Excellences (CoE) within, which specialize in generic areas and facilitate faster turn-around time for development. CoEs deliver their share of a product in the short-term while pursuing development of reusable modules to achieve long-term goals. Conceptually simple, this model may often lead to an avalanche of communication between CoEs related to tracking and Project control when a new product development is undertaken. The alternative is to carry out development forming dedicated product teams to contain this extra communication. But, the organization loses its competitive advantage in the long run due to the absence of “specialization”, i.e., CoE. A CoE transforms cross-product experience into coherent and repeatable body of knowledge. This enables an easily reusable approach to be adopted for new development that eventually results in shorter time-to-market for a product consistently.

2.1. Challenges of a CoE based product development in a distributed environment

Throughout the development life cycle of a product, there are multiple stakeholders at various levels within and outside the organization. Each of these stakeholders would be interested in getting various views of the product development status. For example, an end-customer and the Product Management team view the product as a collection of features. Hence, they would be interested in getting the status of various features within the product during its development cycle. The Release Management team perceives the product as a collection of sequential releases and is concerned about the timeline for each release and the loads within. The development team, on the other hand, would be interested only in the subsystem deliverables for a load within a release. Extraction of the relevant views for the various stakeholders from the project plan is often time consuming and results in redundant information flow in the chain of management. To top it all, geographically distributed development teams add a new dimension to the already complex environment. Management of information flow in the above circumstances is a challenge often not addressed adequately. Tracking of project plans distributed across various geographical locations, tends to become unmanageable when the number of features associated with a product becomes large.

In the discussions to follow, we will depict a suitable framework that will address some of these issues.

2.2. Definition of product, its views and linkages to organization structure

Here, we introduce formalism and define the various entities involved in product development activities. An entity in the context of product development may mean the product itself, the set of features, releases and the various product loads within a release including the CoEs within an organization. These definitions will be used subsequently (Section 3) to explain the planning and tracking of a product using the proposed framework.

2.2.1. Centre of excellence

We assume an organization is structured as a collection of CoEs, i.e.,

\[ O = (\text{CoE}_1, \ldots, \text{CoE}_p). \]  

(2.2.1.1)

Each CoE represents a group having a collection of skilled resources in a specific area. A CoE may either be geographically co-located with other CoEs or distributed. New product development benefits from these
organization lifelines leveraging from the skill sets, the knowledge base and the best practices developed by the CoEs. For example, an organization dealing with development of software for managing a network [4] may have five CoEs such as Fault, Configuration, Accounting, Performance and Security. The “Fault” CoE may be responsible for developing a reusable fault management framework that may consist of protocol agnostic components such as event correlator, alarm browser, fault locator, etc. If the organization has to develop network management software for an optical network, the “Fault” CoE can readily re-use these components (“plug-and-play”) that would have been developed for a Wireless Network Management Software product. Hence, it shrinks development time and improves the time-to-market figure for a new product. CoEs deliver components, which would have been tested thoroughly across various products. New product development benefits from this by suitably mapping (Fig. 1) its requirements to the subset of software components from a framework, thus, reducing the time to market significantly.

2.2.2. Product, releases and loads

We define product as an aggregation of features/requirements, where ‘q’ is the total number of features for this product, i.e.,

\[ P = F = (F_1, F_2, \ldots, F_q). \]  

(2.2.2.1)

This will normally be the view of the product management team for a new product. This may as well be the view of the end users. From the development perspective, each \( F_i \) is decomposed into work in multiple CoEs. It is the responsibility of the product estimation team to arrive at the quantum of work required out of each CoE to accomplish a feature.

As is common, an organization also dedicates resources for planning various releases of a product where ‘r’ is the total number of releases for this product, i.e.,

\[ R = (R_1, \ldots, R_r). \]  

(2.2.2.2)

The scope of a specific release, \( R_i \), is a subset of \( F \), i.e.,

\[ R_i \subseteq F \quad \forall i \in [1, \ldots, r]. \]  

(2.2.2.3)

The temporal ordering between different product releases establishes \( R_i \subseteq R_j \), whenever \( i \leq j \) and \( R = F \).

For a large product, the number of features is large (~200). Accordingly, to ensure finer level of management control, even a specific release, \( R_i \), is divided into multiple loads, \( L_i^j \) (\( i \)th load for \( i \)th release, \( R_i \)), such that,

\[ (a) \quad L_i^j \subseteq R_i \subseteq F \quad \forall i \in [1, \ldots, r], \forall j \in [1 \ldots n_i], \text{ where } n_i = \text{number of loads associated with a specific release, } R_i \]  

(2.2.2.4)

\[ (b) \quad R_i \text{ is divided into load sequence } (L_1^i, L_2^i, \ldots, L_n^i), \]  

where \( k = n_i \), and \( R_i = L_1^i \)  

(2.2.2.5)

\[ (c) \quad L_i^j \text{s are temporally ordered and } L_i^j \subseteq L_i^u \quad \text{whenever, } t \leq u. \]  

(2.2.2.6)

\( n_i \)s in (2.2.2.4) are assumed to be constant throughout the discussion. They are typically decided before the plan based on the duration of a release, need for early testing of the product, etc.

The set of loads for a product throughout its development cycle is defined as:

\[ L = ((L_1^1, \ldots, L_{n_1}^1), (L_2^1, \ldots, L_{n_2}^2), \ldots, (L_r^1, \ldots, L_{n_r}^r)). \]  

(2.2.2.7)

Finally, a set consisting of all the entities as above, related to a product, is defined as,

\[ ES(\text{EntitySet}) = (P, F, R, L). \]  

(2.2.2.8)

2.2.3. Features

A feature is a requirement that is visible to the end customers (e.g., an operator in a telecom world). Product management typically negotiates features with the end customers. Visibility of the feature level schedule to the product management team is extremely important. For products having longer development cycles, a customer normally schedules intermediate checkpoints to ensure logical conclusion of development of a significant subset of the entire feature set (either at a load level or at a release level). Work associated with a feature is decomposed and distributed across various CoEs.

2.3. Phases of an entity

Each entity \( \varepsilon ES \) involved in product development has an associated set of phases with it. The phases of an entity represent major activities that the entity must go through over time.

For a feature, its development life cycle is divided into distinct phases associated with it – normally, Requirement through Testing.

\[ \Phi(F_i) = (\Phi_1^i, \Phi_2^i, \ldots, \Phi_j^i), \]  

(2.3.1)
where ‘f’ is the number of phases in the development life cycle of a feature. Each of $\Phi_i^t$ denotes the $t$th distinct phase for the $i$th feature, $F_i$. Normally, $\Phi(F_i)$ is constant for all $F_i \in F$.

Each phase has a duration associated with it that is represented by

$$\text{Duration} (\Phi(F_i)) = (d_1^t, d_2^t, \ldots, d_f^t),$$

where $d_i^t = \text{Duration} (\Phi_i^t)$ where $t \in [1, \ldots, f]$ \hspace{1cm} (2.3.2)

$\Phi_i^t - s$ satisfy temporal constraints such as,

$$\Phi_i^t \text{ precedes (in time) } \Phi_i^u,$$

whenever $t \leq u$. \hspace{1cm} (2.3.3)

Thus, at any point of time, only one of $\Phi_i^t$s will be active for a feature, or for any entity.

Feature $F_i$ is realized through work items from a subset of $O$ (2.2.1.1), and, consequently,

$$d_i^t = \max\{d_1^t, d_2^t, \ldots, d_p^t\}$$

where, ‘p’ is the number of CoEs and ‘$d_k^t$’ represents the duration of the work for the $k$th-phase of the $i$th-feature from the $k$th-CoE. Each ‘$d_k^t$’ is determined based on the estimate of the work from the feature to the CoE, and remains constant throughout the planning phase.

In the software domain, phases of a feature can be defined based on SDLC (Software Development Life Cycle) and typically consist of Requirements, Architecture, Design, Code and Unit Test (CUT), Integration Testing (IT), System Testing (ST), and Customer Acceptance Testing. Without any loss of generality we will assume for the rest of the discussions, that the domain we are handling is software, where phases of any feature (e.g., $F_i$) consist of the following:

$$\Phi(F_i) = \text{Design, CUT, IT, ST} \hspace{1cm} (2.3.5)$$

Phases for composite entities such as $R$ (including individual loads for a specific release), can accordingly be defined in terms of phases of their constituents. For example, $\text{Duration} (\Phi(L_i^t))$ (jth load for ith Release) can be defined as follows:

$$\text{Duration} (\Phi(L_i^t)) = \{\max(d_k^t \text{ k such that } F_k \in L_i^t), \max(d_k^t \text{ k such that } F_k \in L_i^t), \ldots, \max(d_k^t \text{ k such that } F_k \in L_i^t)\}$$

(2.3.6)

where each of the $d_k^t$s are calculated using (2.3.4).

(2.3.6) has been used (Fig. 3) in the design of the framework in this paper.

3. The problem

Given $O$ (2.2.1.1), $F$ and $P$ (2.2.2.1), $R$ (2.2.2.2) and definitions for each $F_i$ ($i = 1, \ldots, q$), Ref (2.2.2.1) and (2.3.1)), it is important that the following problems be addressed in the context of large product development.

How do we define a framework that will:

(a) ensure near optimal planning/tracking of activities?
(b) support various views of the schedule at a Product, Release, Load, CoE, Feature level?
(c) allow just as much (automated) communication as is necessary between various stakeholders, i.e., communication between CoEs, $O$, and CoE, $O$, is minimal during development phase?
(d) guide in distributing activities of each feature, $F_i \in F$ across $O$, $R$ and $L$?
(e) accomplish all of the above quicker?

(a) and (c) are discussed in Section 4.4, and (b) is explained in Section 3.1.2. A discussion follows in Section 4.1 regarding (d) along with possible future work. Section 4.2 outlines strategies for shortening the time for plan generation and addresses (e).

The framework that we propose here makes no assumption about the location of the CoEs. So, this approach can be applied to geographically distributed CoEs (or teams) as seamless as possible.

3.1. Planning/tracking framework

The framework that we propose here is developed using Microsoft Project (MS Project). While Microsoft Project takes care of most of the scheduling issues, the discipline of organizing interaction between various CoEs in our framework directly enables the following:

(a) Efficient communication between CoEs,
(b) Multiple views (at a load/release/product level) of the product,
(c) Efficient generation of project plan for a large product typically consisting of more than 200 features.

3.1.1. General structure of project plans

We used “Project Sub-project” discipline of MS Project for arriving at the overall product plan. In our framework, the plan typically maps organization structure, $O$, allowing each CoE, in the organization to be treated as a sub-project (Fig. 2). Each of them has a dedicated Microsoft Project Plan (mpp) file that captures the planning and dependencies within a CoE. A global resource file is consolidated at the organization level and all the CoEs reuse that. To have the transparency of the schedule(s) at the feature level, $F$ is represented in a dedicated mpp file with linkages to the feature specific deliverables out of each CoE. Visibility of Load level schedules is brought in by creating a load level mpp file, $L$, with linkages to the tasks in the mpp files representing $F$ and $O$. Communication between the CoEs is facilitated through a protocol as defined in the subsequent section.

---

1 Microsoft is a registered trademark for Microsoft Corporation, USA.
Resources (as an mpp file) attached to multiple CoEs are also reflected in the background in Fig. 2. The overall structure of the project plan is divided into following components:

(a) **Product mpp File**: Representing entities such as \(P\), \(R\) and \(L\).

(b) **Feature mpp File**: Representing \(F\).

(c) **CoE mpp File(s)**: Representing plans for \(O\).

(d) **Global Resource mpp File**: Shared resource plan.

We have integrated the plans for \(P\), \(R\) and \(L\) into one for convenience, though this is not necessary. A separate plan is drawn at a feature level (Feature File) that has direct linkages to Product File as well as CoE Files. The global resource file contains details of resources – human as well as others such as hardware/software, necessary for product development.

### 3.1.2. Linking multiple views of a product

As explained in Section 2.2, the framework should be capable of integrating different views of a product during its development cycle. Specifically, it should abstract out the load view (of interest to developers/release management), release view (demanded by release management), feature view (for end customers and product management), and the CoE view for development.

The feature view integrates schedules of all the features into one view. Given \(O\) and \(F\), \(\Phi^f_i(2.3.1)\) is dependent on pieces of work from various CoEs. In the plan for CoE\(_j\), placeholder tasks for \(F_i\) are retained in the Export block (Fig. 5) that indicates contribution of CoE\(_j\) towards \(F_i\).
Tasks representing $\Phi$'s in the feature plan, are linked to these tasks from the CoE plans. This eventually connects the feature and the CoE views (Fig. 3) for a product plan.

It is evident from Fig. 3 that the bulk of the work (Design, CUT) is done at CoE level ($CoE_m.mpp$). The Feature file ($Feature.mpp$) is a consolidation (typically, milestones in MS Project terminology) of similar activities (e.g., Design, CUT) of all the features ($F_i$s) from the various CoEs ($CoE_m.mpp$), thus offering feature level view of the product. The Product file ($Product.mpp$) contains tasks that represent various releases ($R_m$) and loads ($L_m^a$). Various features ($F_i$s from the feature file) contribute to a load ($L_m^a$) in the product file that has predecessor linkages to different feature phases in the feature file (Fig. 3).

As soon as a CoEx finishes its phase (e.g., Design, CUT) for a feature, $F_i$, it announces (through predecessor linkages, label 1 in Fig. 3) the same to the appropriate phase (respective Design, CUT from all CoEs) of $F_i$ in the feature file. Different $F_i$s (from the feature file) belonging to a specific release ($R_m$) and a specific load ($L_m^a$), next, broadcasts (label 2, Fig. 3) the same to the corresponding phases (Design $\rightarrow$ Design, CUT $\rightarrow$ CUT, etc.) of $L_m^a$ in the product file. When the product file receives completion signal for a phase (e.g., Design, CUT) from all the features constituting load, $L_m^a$, it enters the IT (integration test) phase for $L_m^a$, and announces (label 6, Fig. 3) the same to all the CoEs, on which, they enter the IT (e.g., defect fixing) phase. Once IT phase is completed, CoEs announce (label 7, Fig. 3) completion events to the product file, which in turn, broadcasts the same (Label 8, Fig. 3) to all the respective features, constituting $L_m^a$, in the feature file. Product file also signals start of ST (label 9, Fig. 3) to all the CoEs at this stage, and CoEs enter ST phase. CoEs indicate (label 10, Fig. 3) completion of ST to the product file that, in turn, communicates the same to the feature file (label 11, Fig. 3) completing the cycle.

Given the dynamics of automatic consolidation from the individual updates of the project plans, it is clear that, at each stage, by virtue of the structure of the project plan, information remains largely up-to-date and consistent. Most often, all the tasks in the feature file and many in the product files are mere milestones, thus, obviating the overhead of tracking a plan by the product owner. Also, the CoEs are independent having well-defined communication protocol (discussed in Section 3.1.3), and so can be operated independently by the CoE managers. This greatly reduces the interaction/confusion that may be prevalent in managing distributed development of a product.

3.1.3. Structure of a plan to facilitate communication between entities

The internal structure of a plan (for an entity) plays a crucial role in pulling up the avalanche of communication between teams (CoEs). If the protocol for inter-entity communication is captured in a structured manner in the individual entity (mpp file) plans, dependencies can be easily understood across teams, thus leading to easier communication.

Keeping the above in mind, a plan is structured to consist of two blocks – namely, the “Internal” block and the “Export” block (Fig. 4). The “Internal” block contains activities (tasks in mpp file) internal to the entity (typically CoEs). The “Export” block contains typically milestones that are dependent on tasks in the “Internal” block. A task (milestone) is placed in the “Export” block to announce this milestone to any other plans that may depend on this.

3.1.3.1. Structure of the “Export” block. A CoE (including the product file) communicates with other CoEs through the tasks marked as “CoE Exported Milestones” in the “Export” block (Fig. 5) in the plan. As the plan evolves, each CoE needs to agree upon its mutual dependencies on other CoEs and announce the same in the “Export” block in the plan. A dependent CoE (mpp) file, in turn, establishes dependencies on these tasks in its internal block if required.

In order to get a view of the feature deliverables out of a CoE (mpp), the feature milestones are made visible (marked “Feature Exported Milestones”) in the Export block. Subsequently, these milestones are consolidated as predecessors to corresponding features in the Feature file.

The export Block may contain additional structures if the plan so demands. However, the above two blocks within the “Export” block clearly facilitates any communication that may be necessary between geographically distributed CoEs having their own plans.

3.1.3.2. Structure of the “Internal” block. The “Internal” block is the actual plan for a CoE. Typically, this is a monolithic (project) plan, which captures schedules for a CoE’s internal modules. The internal modules within a CoE plan together realize features scheduled for a specific load/release.
4. Product plan using the framework

In a real project environment, the plan for a large product is organized as master and sub-project plans using MS Project. The overall plan is then supported by a resource mpp, which represents available human resources and materials in an organization. CoE managers create individual CoE plans based on product requirements or features. These plans are then integrated into a Feature mpp, and subsequently, consolidated as a Product mpp. For a large product, this offers greater control and parallelism to CoEs, even if they are geographically distributed.

An important step towards planning a large product involves content definition for the \(j\)th load within the \(i\)th product release, i.e., \(L'_{j} (2.2.2.4)\) for a given \(F (2.2.2.1)\). The framework uses (phase) synchronization points \(2U(L'_{j})\) for \(L'_{j}\). As is clear from (2.3.6), these synchronization points may introduce idle slots. A feature that finishes early has to wait at a synchronization point (typically at a phase cross-over) till all its “fellow” features \(\in L'_{j}\) complete their parts corresponding to that load phase. Hence, careful selection of features (load content) for a particular load plays a crucial role in optimizing development plan for a large product.

4.1. Defining load content for a release

It is a development imperative that the load content (set of candidate features for the \(j\)th load within the \(i\)th release, i.e., \(L'_{j}\)) is chosen carefully so as to maximize the number of independent tasks in a load. Following (2.2.2.4), a release contains several loads within it. For a new product, the development organization has a reasonable say in scoping features across loads within a release. Given that, a systematic approach can be adopted to ensure optimal resource utilization that may result in improved schedule for a release. This is typically an iterative process and the ultimate goal is to arrive at the load content such that the variance is minimum, i.e.,

\[
[\sigma'^2] \text{ is minimum, } \forall \ d_{i,j} \text{ such that,}
\]

\[
t \in [1, f], l \in [1, p] \text{ and } F_k \in L'_{j} \forall j \in [1, r], \ i.e., \text{all loads for the } i\text{th-release}
\]

where \(f = \text{number of Phases for a feature (2.3.1)}, \ p = \text{number of CoEs in an organization (2.2.1.1)}, \ n_1, \ldots, n_r\) and ‘\(r\)' are constants (2.2.2.4).

To arrive at an optimal load contents for a release, one has to consider factors such as dependencies between features, and ensure near equal distribution of efforts across loads. Within a load, inter-CoE idle slots (i.e., minimize \(\sigma'^2\)) have to be minimized to the extent possible, since a CoE may finish its deliverables earlier than other CoEs. Even before one gets into detailed plan, these issues are to be resolved. Otherwise a re-plan may be necessary at a later stage, which is often costly for a large product.

4.2. Automated generation of plans

Given the enormity of the number of features, and the challenge of getting a detailed view of the product at a feature level, it is almost impossible to come up with a plan manually. The feature file, for example, captures the status of all features based on the progress of each
CoE. The number of links that will be necessary to maintain a feature plan is too large to be generated manually.

Let, \( q \) be the total number of features for a particular release of a product, \( f \) be the cardinality of the set, \( \Phi(F_i) \) (2.3.1). If the number of CoEs is \( p' \), the feature file, in the worst case, will contain \( O(q \cdot f \cdot p') \) (external) predecessor links. Assuming \( q = 200, f = 4 \) (2.3.5), \( p = 10 \), this results in \(~8000 \) (external) predecessor links, which is virtually impossible to generate manually. Similarly, the product plan (mpp) will have linkages to the feature plan (mpp) for every feature and it adds equally large number of linkages. Necessarily, we need to go for automatic generation of plans wherever possible.

The structure of the Project Plan framework that we presented facilitates automatic generation of the various links (dependencies) between the plans and the generation of the basic plans itself. The framework also offers scalability with respect to addition of new Features for a Release, and addition of new CoEs in an Organization. There are built-in macros to facilitate generation of the feature plan from the load contents in the framework. It also automatically creates placeholder tasks for all the features across the various phases of development in the Export Block of the CoE plan (Fig. 5, marked under “Feature Exported Milestones”). The manager has to focus on developing the project plan (“Internal Modules” in Fig. 5) for the relevant features internal to the CoE. Finally, one has to link the tasks within the Internal Block to the specific features in the Export Block of the plan as highlighted in Fig. 5.

4.3. Integration of plans across CoEs

Once the CoE plans are ready, integration of these plans with the product plan and the feature plan begins. Over and above the regular debugging issues associated with a project plan, one encounters additional problems due to Circular Dependencies (CD) of tasks between the CoE plans during the integration phase. CDs may occur due to lack of common understanding of inter-CoE communication protocol, or due to oversight in translating load contents into the CoE plans.

Suppose, \( F_i \in L_1^m \) and \( (F_i \in L_2^m \) and \( F_i \notin L_3^m \) where \( m < n \) (2.2.2.6), according to the load contents. If within a CoE plan, it is assumed otherwise, i.e., \( F_i \in L_1^m \) and \( F_i \notin L_3^m \), the oversight results in a circular dependency (violation of temporal ordering of features) in the framework. Tasks (Fig. 3) under \( L_3^m \) will have \( F_i \) as the predecessor in the product (mpp) plan, which results in the temporal ordering of \((F_i,L_2^m,F_3,L_1^m)\). The oversight in the CoE plan leads to a situation where the temporal ordering becomes \((F_i,L_2^m,F_i,L_1^m)\) resulting in causal violation.

The framework facilitates quicker resolution of circular dependencies in a plan by virtue of its structure.

For a large plan, the task of debugging circular dependencies is complex. The guideline that we propose to resolve such issues is as follows:

1. [Network diagram test] Open the plans under MS Project and expand all tasks under each plan. Click on the Network Diagram view in the product plan (mpp). If a plan does not contain CDs, this operation will result in a well laid-out Network Diagram. Otherwise, due to the presence of CDs, the tasks in the plan get juxtaposed and the diagram will not be clear.

2. If (1) indicates the existence of CDs, go to Step-3 in order to identify the ‘suspect tasks’, else go to Step-5.

3. In each plan involved in a CD, identify the set of ‘suspect’ tasks as follows:
   a. [External CD] Filter tasks whose predecessors are outside the plan (\( P = “\text{External}” \)), and successors are inside the plan (\( S = “\text{Internal}” \)). These tasks may potentially be involved in a CD. Mark them as ‘suspect’
   b. [External CD] Filter tasks such that \( (P = “\text{Internal}” \), \( S = “\text{Internal}” \)). Mark them as ‘suspect’
   c. [Internal CD] Filter tasks such that \( (P = “\text{Internal}” \), \( S = “\text{Internal}” \)). Mark them as ‘suspect’

Please note by virtue of the structure of the framework, there will not be any task in a plan with \( (P = “\text{External}” \), \( S = “\text{External}” \)) as a configuration. Also, CDs due to (c) will be entirely within a CoE plan and would have most likely been debugged by CoE manager before coming to this stage.

4. Take one \( (T) \) of the “suspect” tasks obtained from Step-3. Examine its predecessors \( \{P_1,P_2,P_3,\ldots\} \). Take \( P_1 \), and remove it from the plan. Observe if CD disappears (Network diagram test). If so, \( (T,P_1) \) are adjacent nodes in a CD. If no, restore \( P_1 \) and examine \( P_2 \), and so on.

5. Exit

While scheduling features during the development cycle, some features may require splitting across loads. This may sometimes lead to improper assumptions about the scope of each load across CoE plans, and a circular dependency error may sneak in.

4.4. Distributed tracking and control

We recommend that the overall plans be organized as a “Master-Project Sub-Project” combination under MS Project. A CoE manager operates only on his or her CoE (mpp) plan, which is a sub-project file. Periodic tracking or modification of the tasks by a CoE manager is, thus, just confined to his or her respective CoE plan(s). The Master Project represents the consolidation of all the CoE/Feature/Product plans. In summary, the framework does not bring any additional tracking overhead to any of the stakeholders in the system. On the other hand, various levels of abstraction of the
Product status are generated automatically by the framework with no additional effort.

We used the “Workgroup” feature of MS Project for automated tracking of all the plans. Using this feature, tracking-requests for tasks within a time window can be sent to the task owners through emails automatically. Status updates, which come as reply emails from each owner, are automatically consolidated by MS Project itself.

4.5. Requirement volatility and the framework

The proposed framework ensures a high degree of immunity to the plans against requirement volatility. For a large product, since the development time is long, changes in requirements are inevitable. During the development cycle, features may have to be added, re-scheduled across loads, dropped, or their scope may undergo changes. If a feature is rescheduled to a different load, the CoE plans have to just change the trigger tasks to the new load. Everything else remains unchanged. Only the product plan needs to link the feature in the feature file to the new target (load) in the product plan appropriately. If a feature is dropped, we may merely re-assign the feature efforts to “0” in the CoE plans: the feature and the product plans remain unchanged. When a new feature is added, the CoE plans require replanning the most as is expected, while the Feature plan needs to setup linkages (achieved using MS Project macros automatically) to the Export Blocks of the CoE Plans. The product plan links the new feature in the feature file to the target load in the product (mpp) file. Similarly, in the event of a change in the feature scope, the CoE (mpp) plans undergo changes, but the product and the feature plans remain unaffected.

4.6. A case study using the framework

The framework that we proposed was used to arrive at the project plans for the development of a large Networking Product. There were 209 features in the system to start with that got effectively reduced to 200 by addition and deletion of some features. The development spanned over eighteen months with five CoEs and a collective effort of 60+ person-years. The project plans were mapped into the organization structure using the framework. The Feature Plan and the Product Plan consolidated various views, which are of interest to the Product Management and the Release Management teams. Generation of the templates for the CoE plans, Feature Plan and the Product Plan was automated using the macros that we developed under MS Project. Roles were defined within the organization, where the Release Manager was responsible for maintaining the Product Plan, and the Feature Plan and the subsystem managers were responsible for the CoE plans.

During the development of the plans, automation significantly speeded up the time for generation of the plans. Fig. 6 compares the plan-generation time using manual (estimated) and automated (by the framework) approaches. Data for the manual mode was estimated based on the fact that the time taken will be proportional to the number of links \( (q.4.5) \) (Section 4.2, \( f = 4, p = 5 \)). We assumed an average of 1.5 min (highly optimistic) for setting up a task link manually between the Feature Plan and a CoE plan. It is evident from this figure that beyond a certain threshold for the number of features, it is impractical to generate the plans manually if the plans are to provide diverse views to the various stakeholders.

Plans were tracked at the CoE level by the respective subsystem managers on a weekly basis. The Release Manager had to just open the Product Plan under MS Project for automatic consolidation.

Tasks within the Feature Plan were linked to the tasks in the Product Plan and the tasks in the Export Blocks of the CoE plans. It comprised of a series of milestones for each feature and updation of these milestones was automatic whenever there were changes in the CoE plans and the Product Plan. To introduce more
automation in the system, workgroup feature of MS Project was setup in the system, which automated the tracking mechanism at the CoE levels. Various views of interest to the Release Manager and the subsystem managers were readily available from the Product Plan, the Feature Plan and the CoE plans.

5. Conclusion

We have proposed a new project management framework that facilitates development of project plan for a large product in a geographically distributed environment. The methodology developed here applies in the context of product development organizations, which are structured as a collection of CoEs. The framework makes no assumption about the location of the CoEs, and so, they need not be geographically co-located. The framework is capable of integrating diverse views (Feature/Load/Release) of a product plan while enabling coherent, consistent presentation of these views seamlessly to the various stakeholders such as end customer, product management team, development team, and the higher management team. The structure of a product plan (Internal/Export blocks) using the proposed framework ensures minimal coupling and organized communication between the CoEs, offering a great degree of immunity against requirements volatility. It also facilitates easy debugging, as discussed in the paper, during the development of the project plan itself. The use of workgroup feature obviates manual tracking. The project management framework, thus, eliminates any geographical barrier between the CoEs, enabling organizations to capitalize on distributed skills for a successful product development.

While we attempted to solve the problems of arriving at and maintaining a large product plan, the issues regarding optimal load contents plan remain largely unanswered. An optimal load contents plan will minimize idle slots between CUT complete and IT within a load (Fig. 3), thereby solving (4.1.1), which may be the future area of work.

The examples that we considered in this paper were taken from the Software domain. One can, however, apply the same methodology to other disciplines as well.

Finally, we experimented this framework for developing plans for a large software product having around 200 features, which we successfully deployed in our organization.

Acknowledgements

We thank Surendra Saxena, Former Director, OMC Group, Lucent Technologies (India) Ltd., for his continued support and co-operation in making this happen. Thanks are due to our colleagues Sukirshna Dutta, Gopalkrishna Holla, Nitin Acharya, and Nireesh Nagaraj who used this framework to come up with their CoE plans.

References