

A Model for Platform Development

For many software and firmware products today, creating the entire architecture and design and all the software modules from the ground up is no longer feasible, especially from the point of view of product quality, ease of implementation, and short product development schedules. Therefore, the trend is to create new product versions by intentionally reusing the architecture, design, and code from an established software platform.

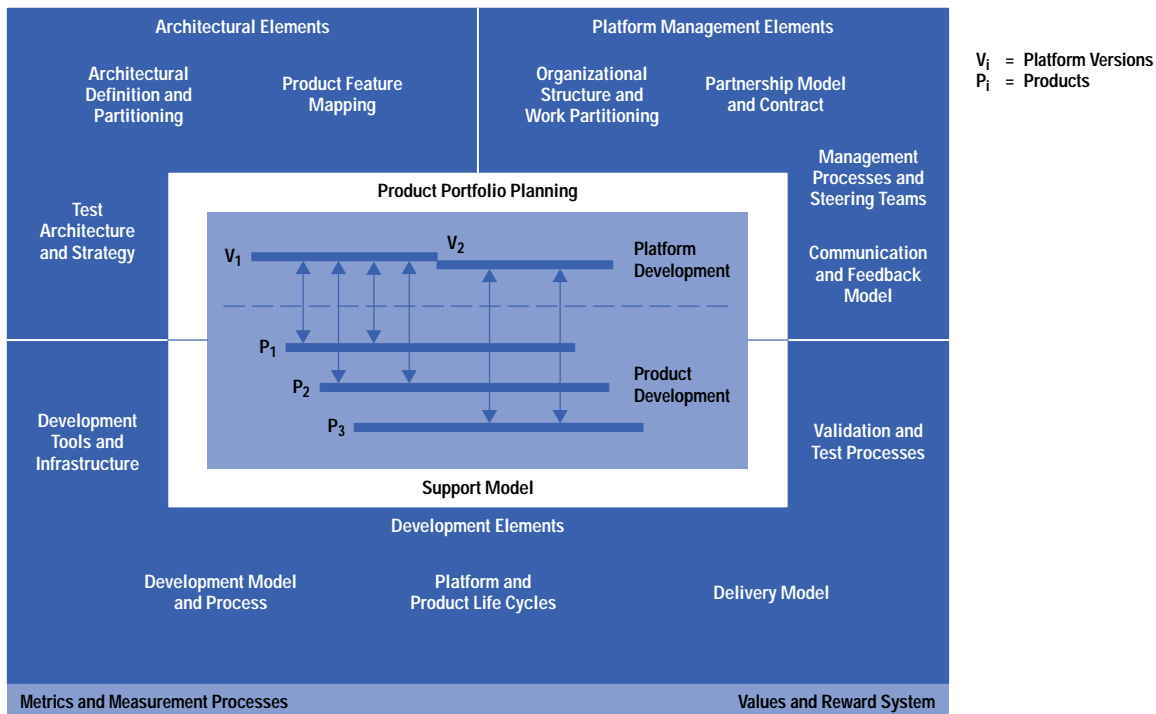
by **Emil Jandourek**

HP's software initiative program has been working in partnership with product development organizations in Hewlett-Packard for almost five years. Its goal is to help take software and firmware development off the critical path of new product introductions and transform HP's software and firmware development capability into a competitive advantage. Through our work we have observed and participated in the application of many different strategies, all aimed at raising an R&D team's collective ability to build software and firmware that meets the overall market requirements, including functionality, usability, reliability, performance, supportability, and time-to-market goals.

Several patterns have emerged that many HP organizations are successfully using to elevate their software and firmware development capability. One pattern corresponds to a set of operational practices that we call the *platform development paradigm*. The software initiative program has created a conceptual model for platform development (see Fig. 1) which builds upon HP's product development experience and integrates many of HP's best practices in software development. The individual elements of the model are closely tied to the technical and management systems used in the company and have been validated through actual team experiences in developing new products.

Since the platform development model is conceptual, it is used as a framework for determining the elements that an organization needs to invest in to attain a competency in platform development. The software initiative program works with product development organizations to identify the areas of the model that are applicable to a given organization's situation and works with the organization to customize the model accordingly. The resulting instantiation of the model yields processes tuned to the specific needs and requirements of the particular development organization, leading to a new level of development capability. Organizations within HP that have established a competency in platform development have

Fig. 1. Major elements of the platform development model.



significantly reduced their time to market, improved operational efficiency, and become more responsive to the needs of their customers. These gains are accompanied by improved business results.

The following are brief descriptions of the elements of the platform development model shown in Fig. 1.

- **Product Portfolio Planning.** This element defines the strategic relationship between the platform and all product versions to be released over a multiyear period. It identifies the key business drivers and sets the overall goals, direction, priorities, and parameters of the platform strategy.
- **Architecture.** This group of elements includes:
 - Architectural definition and partitioning of the major functional and technology subsystems.
 - Product feature mapping, which identifies appropriate subsystems and component modules used in the implementation of each feature (i.e., translation of customer needs to product features to specific platform or product modules)
 - Test architecture and strategy, which define the overall structure and methods for verification and validation to ensure necessary quality levels in the final product.
- **Platform Management.** This group of elements includes:
 - Organizational structure and work partitioning, which defines the organization's operating model at an abstract level (e.g., reporting relationships and team organization)
 - Partnership model and contract, which provides the generic framework for instantiating the operating model between platform and product teams (e.g., interdependence between teams and expectations for their working relationships)
 - Management processes and steering teams, which define how the product portfolio plan is created and how its execution is managed
 - Communication and feedback model, which defines the timing and content of the information that flows between teams.
- **Development.** This group of elements includes:
 - Platform and product life cycles, which define the major phases, with goals, activities, and deliverables for both the platform and products
 - Development model and process, which specify the processes followed for the creation and enhancement of a module through its integration into the final product
 - Delivery model, which defines how platform components and subsystems are delivered for use within products
 - Validation and test processes, which define the specific quality criteria and test procedures used throughout the product and platform life cycles
 - Development tools and infrastructure, which provide a common development environment and processes for platform and product work (e.g., procedures and tools for creating, storing, finding, building, and testing components).
- **Support Model.** This element defines the mechanics and logistics of how individuals and teams get help when using platform components.
- **Metrics and Measurement Processes.** This element defines the means by which progress and results for each of the other elements are monitored to ensure achievement of business goals.
- **Values and Reward System.** This element integrates and aligns the organization's values and culture with its performance evaluation and reward mechanisms to support the other elements of the model and thereby achieve platform, product, and business goals.

The remainder of this article describes the key elements of the model in greater detail, including the deployment and use of the elements, anecdotes about their implementation, and finally, HP's experiences with the model. The use of the word "software" throughout this article refers to both software and firmware.

Definitions and Background

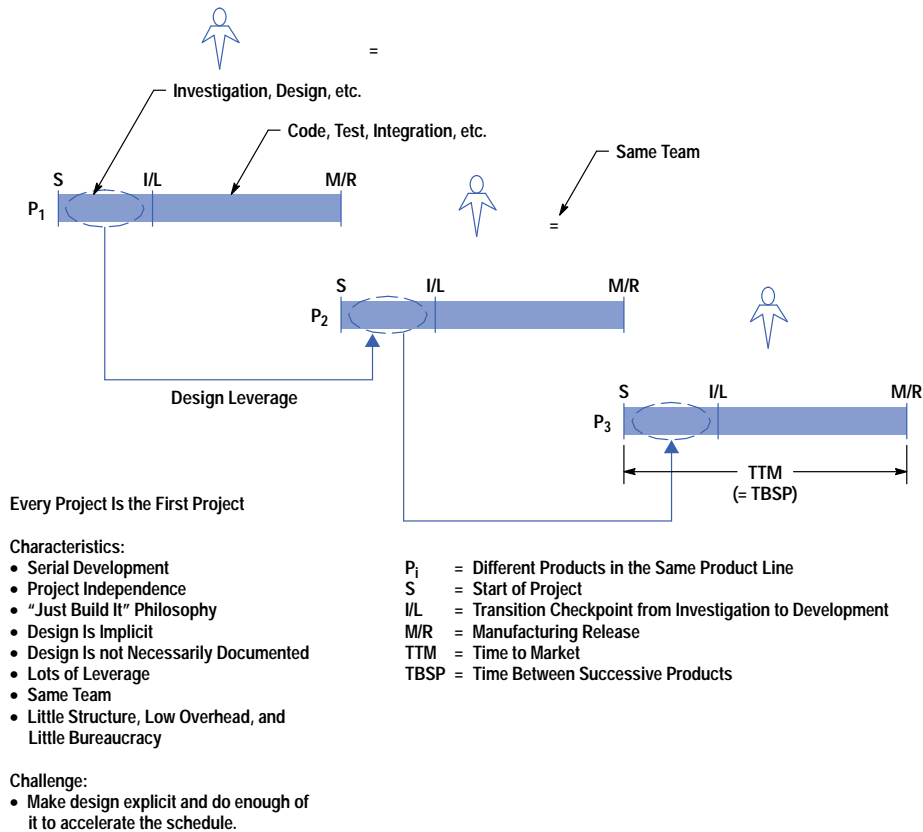
For HP and many other high-technology businesses, the evolution of product development organizations parallels that of a company's business. The character of a business changes as its products evolve, mature, and expand their market penetration beyond the innovators and early adopters.

This technology adoption life cycle has implications for how an organization develops its products.¹ An organization's first product for a new, emerging market is often an experiment aimed at validating a product concept and getting feedback to help shape its evolution. Consequently, the first product is often incomplete and may in fact be a cleaned-up prototype. Successful market introduction and subsequent demand for the product inevitably lead to plans for follow-on products.

Paradigm I: Serial Development Projects

Development during the early stage of a new product's life cycle is characterized by a series of independent projects (see Fig. 2). A "just build it" mentality often drives the first few products because of uncertainty about the market acceptance of the product. From a software development perspective, the product's architecture and design are often implicit and poorly

Fig. 2. Paradigm I, serial development projects.



documented. Little structure and formality work reasonably well for small development teams as long as there is continuity between the initial product team and the teams that develop subsequent products. In fact, very often the initial team and the teams for follow-on products are the same. This continuity of individuals and teams enables both design and code leverage between projects.

In paradigm I, the time to market (TTM) is defined as the time between the start or initial staffing of the project and its release to customers. Organizational learning and leverage between any two successive projects can reduce the TTM for the latter project. Thus, if a similar amount of functionality is contained in both projects one expects the TTM for project $N + 1$ to be less than the TTM for project N . Ideally, the bulk of the effort invested in a latter project is directed at those value-added, differentiating features that are visible to customers.

An organization can choose any number of different development methodologies or life cycles for its development effort. Within HP, many organizations are adopting an evolutionary delivery approach as opposed to a waterfall model (see [Article 5](#), [Article 3](#) and reference 2). In fact, even those using a waterfall model have modified it to support increased concurrency and reduce the impact of a reset at any stage.

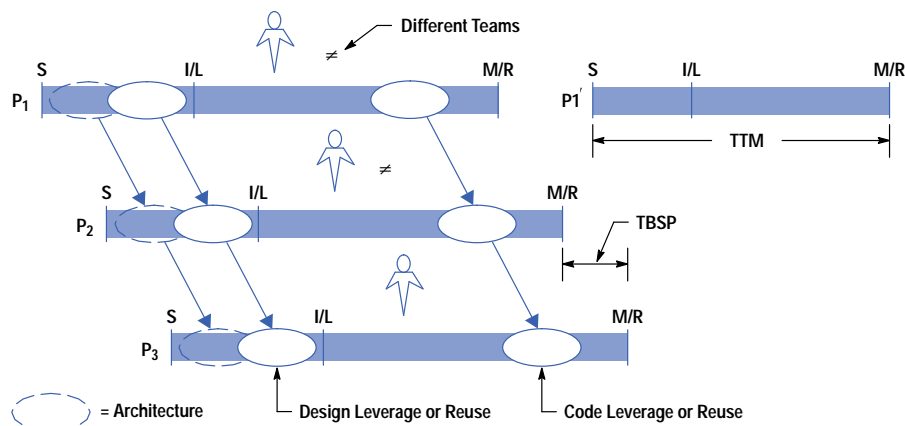
The time between successive products (TBSP) is calculated by subtracting the release date of the earlier product from that of the later product. This measure is referred to as time to market prime (TTM') by some organizations. In the case of serial independent projects TBSP is equivalent to the TTM of the last product.

Paradigm II: Multiple Parallel Projects

When a business finds increased market acceptance of its products and has market penetration beyond the innovators and early adopters, it is common for customers to demand follow-on products with shorter intervals between them. The pressure to do faster product releases and thereby cut TBSP almost invariably results in a shift within the product development organization to multiple independent projects. We call this paradigm II (see Fig. 3). This situation usually results in different teams working in parallel to build closely related products.

In paradigm II, the TTM does not necessarily change, but the TBSP shrinks because of the overlap between projects. Customer demands for frequent product releases and consistency within a product family put pressure on the product development organization to achieve an appropriate level of consistency across products and shrink both TTM and TBSP. The potential to reduce the TTM for follow-on products exists if there is a high degree of leverage from previous products.

Fig. 3. Paradigm II, multiple parallel projects.



Only One First Product per Portfolio (Customer View)

Characteristics:

- Parallel Development
- Project Independence
- Local Optimization
- Products Look Similar
- Design Is Probably not Explicit
- Different Teams, Some Leverage
- Architecture Benefits Accrue to Follow-on Projects

P_i = Different Products in the Same Product Line

S = Start of Project

I/L = Transition Checkpoint from Investigation to Development

M/R = Manufacturing Release

TTM = Time to Market

TBSP = Time Between Successive Products

Challenge:

- Make leverage or reuse of both design and code happen predictably.

Since leverage fundamentally looks into the past for existing assets to draw upon, there is no guarantee that the software found will not require extensive modifications to work for the new product. Thus, the benefit of leverage is subject to an inherent limitation and in the worst case may be negative (i.e., when the cost of leverage exceeds that of a new implementation).

A much greater reduction in TTM can be achieved if extensive reuse is possible. Reuse fundamentally looks to the future and orients development around what follow-on products will require. Since reusable software components are designed with the future in mind, they can be plugged into new products without any modifications. This is known as black-box reuse because the component user's primary concern is with the external behavior and interfaces and not with the internal details of the component. The practices of leverage and reuse form two ends of a continuum in terms of benefit to recipient project teams. In general, the benefits for project teams that reuse components are greater than those that leverage components. However, components that are not specifically designed for reuse typically need to be modified and hence end up being leveraged. There are significant differences in the development processes used to build reusable components.

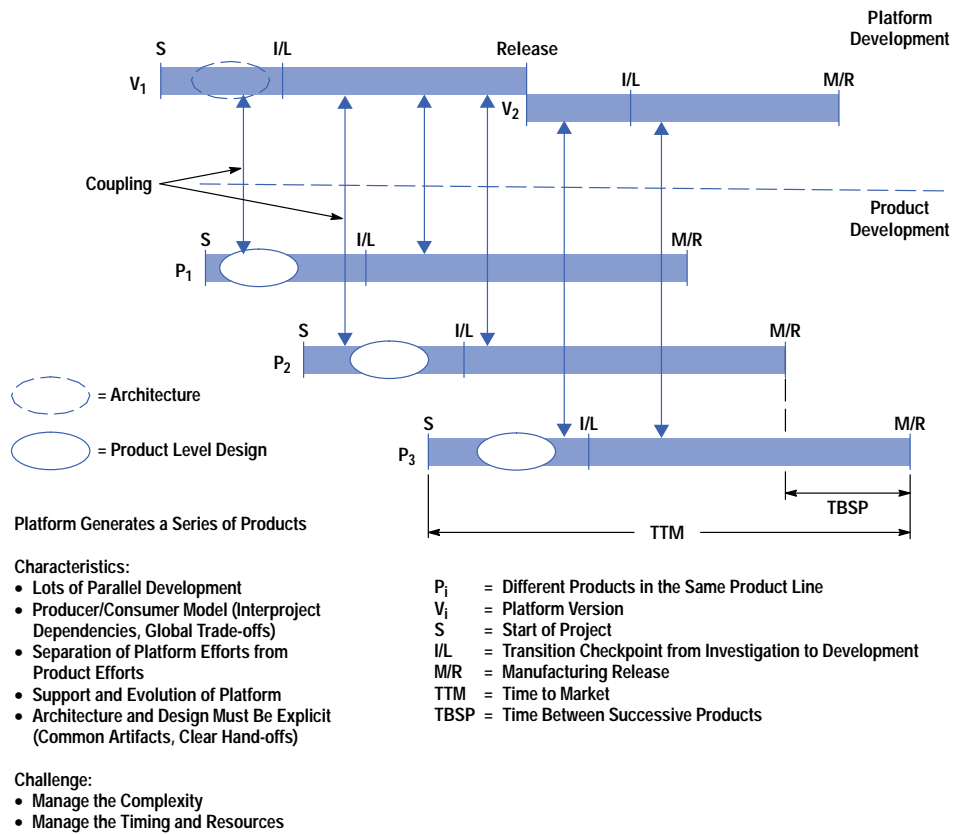
The challenge for organizations doing multiple independent but related projects in parallel is to make the practices of leverage and reuse happen predictably across projects. These practices can happen at multiple levels, from the sharing of architecture and high-level designs to object code and test vectors. The larger the granularity of work shared, the greater the impact on reducing project TTM. For example, reusing a complete error-handling subsystem will reduce project effort more than reusing just a few selected error handling routines.

In paradigm II, individual teams usually have dedicated project managers and architects for each product. This configuration provides each team with a large degree of independence and autonomy. At the same time it can also make it difficult to coordinate the sharing of work between teams. Many organizations find that their predominant mode of operation closely follows paradigm II. For businesses with different product lines there may be several sets of independent projects underway at any given time.

Paradigm III: Platform Development

Businesses that have firmly established a presence for their products in the marketplace, have moved beyond the early adopters, and have achieved a deep customer and product understanding may consider moving to platform development, paradigm III (see Fig. 4). Platform development is essentially an extension of paradigm II, where the common elements within a product family are factored out and developed once. The essence of paradigm III is to pull out those product elements, features, and subsystems that are stable and well-understood, and that provide a basis for value-added, differentiating features.

Fig. 4. Paradigm III, platform development.



A platform is different from a reuse library in that it has a cohesive, underlying architecture. The exact composition of the platform for any given product family can range from a complete product framework to a collection of subsystems to sets of individual components. The platform's contribution to individual products can vary from 10% to nearly 90%, either in terms of code or development effort. The exact amount and form of the contribution depend on the specific needs of each product family. Products developed using the features and pervasive structures (e.g., error handling and GUI standards) resident within the platform have a much shorter TTM.

The shift to platform development takes the effort an organization normally puts into product basics and reduces it through reuse. Although new functionality and features can be provided by either platform or product software, in cases involving a large degree of uncertainty new features are usually implemented as part of the product. Once the new product functionality stabilizes and is accepted by the marketplace, it can be migrated into the platform. Thus, it becomes available to subsequent product development efforts.

The net result of implementing paradigm III is a reduction in the TTM for individual projects. This reduction coupled with the parallel development inherent in paradigm II allows organizations to shrink their TBSP. This also enables better market responsiveness, and not surprisingly, in mature businesses a whole series of platforms may be developed to support different product families.

Examples

The following two examples serve to illustrate the power of platform development. In the consumer electronics world, Sony Electronics Inc. is a large producer of handheld portable, radio and cassette players. In fact, Sony makes over twenty different Walkman® stereo radio and cassette players that it sells in the United States. Close examination of these products reveals that only a few underlying cassette mechanisms and cases are used for the entire product family. These mechanisms and packaging constitute Sony's platforms, and they enable Sony to generate an assortment of products targeted at a broad spectrum of customer needs extremely rapidly. The incremental investment needed for Sony to bring out a new model is small because of the large amount of reuse offered through its platforms. There are numerous other examples like this in the consumer electronics markets.

Within the computer networking market, HP offers a product called HP OpenView, a software product used for managing complex networks. In HP OpenView's case, parts of the software form a platform that HP's customers use to build network management applications. In addition to offering HP OpenView to other network management vendors, HP also markets its own suite of HP OpenView-based network management applications. The HP division responsible for OpenView produces additional network management applications in under half the time required for a ground-up implementation. This represents a TTM reduction of over 50%. Third parties working with HP OpenView also experience similar levels of effort

and time savings. Unlike HP OpenView, which is sold and used external to HP, most development labs are producing and using platforms internally to provide the foundation for individual product lines.

Changing an organization's development paradigm to platform development is nontrivial and requires a significant investment. Richer, more robust, and more competitive products along with TTM reductions of one third to one half are not uncommon. Coupling the use of a platform with doing multiple products in parallel results in significant reductions in TBSP. The lower limit for TBSP is the rate at which the market can absorb new products.

Platform Competency Model

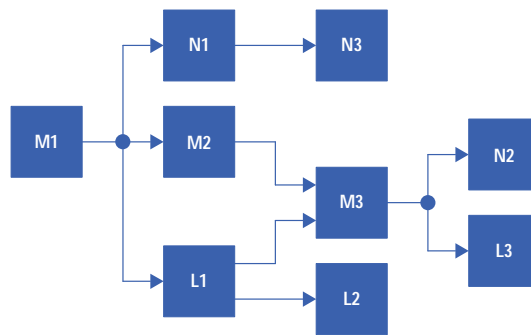
The purpose of the platform competency model is to depict the core elements that make up an organization's software development system (see Fig. 1). Each individual element of the model addresses a particular aspect of how an organization's development system works. The model collectively represents the overall operating model for software development within an organization. At the same time the model is holographic since each of the elements contains references to aspects of the other elements. As a result of this, the model is not amenable to hierarchical decomposition. This will become apparent as each element is reviewed.

Product Portfolio Planning

The heart of the platform competency model centers around what the business requires of the teams developing software. A product development organization is constrained by the dynamics of the marketplace and the competitive environment within which it participates. The combined market and competitive forces determine accepted operating ranges for investment, time-to-market goals, product specifications, ongoing support, and so on. These constraints put limits on development organizations and establish acceptable bounds for TTM, TBSP, and R&D resource efficiency (e.g., engineering years/product).

The result of integrating these constraints and high-level business goals is articulated as part of a business plan. The business plan includes a product vintage chart, which shows target release dates for individual products over a multiyear period (see Fig. 5). Apart from their use in business plans, product vintage charts are often supplemented by a set of product lineage charts showing the hereditary relationships between products and their constituent components. Fig. 6 illustrates the structure used for a traditional lineage chart and Fig. 7 shows a platform-based lineage chart. Separate lineage charts are often constructed to highlight different components of a product (e.g., electrical circuits, software and firmware, industrial design, mechanical assembly, etc.). Lineage charts are also used outside of development labs to depict the evolution of marketing, training, and support materials.

Fig. 6. An example of a lineage chart that might be used for the software in different versions of the workstation products shown in Fig. 5.



Notes:

1. M, N, and L are the same products as shown in Fig. 5
2. Typically, the arrows are annotated with software subsystems or component names
3. Branches generate multiple versions of a software component.

The key in planning a product portfolio is deliberate, systematic attention focused on developing a product lineage chart that effectively addresses the organization's product needs. The software version of the lineage chart sets forth what can be accomplished as a result of the organization's platform strategy. Together the product vintage chart and lineage chart provide the framing for the product portfolio plan (see Fig. 8). The plan establishes the targets for the amount of contribution that a platform makes to individual products. This is frequently expressed as both a target effort savings and a target reuse percentage (i.e., the platform will reduce product investment by X engineering months and will provide Y% of the final product code). It also sets forth the goals for TTM (time to market) and TBSP (time between successive products) for an entire product family.

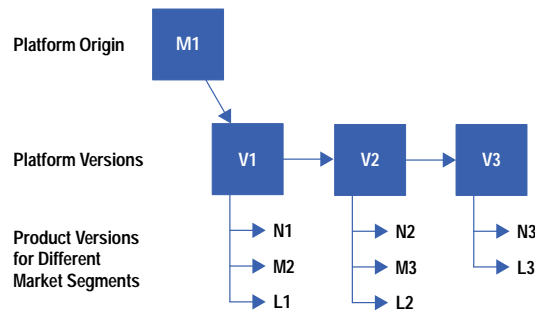
Fig. 5. An example of a product vintage chart that might be used for workstation products.

Target Market Segment*	Products							
High-End (e.g., High-Performance Servers)			N1			N2		N3
Midrange (e.g., Servers and Workstations)		M1	M2		M3			
Low-End (e.g., Low-End Workstations)				L1		L2		L3
	Spring	Fall	Spring	Fall	Spring	Fall	Spring	Fall
	1992		1993		1994		1995	
	Product Release Dates							

* The market segment also determines the product's price.

The product portfolio plan also includes a statement of overall goals, direction, and priorities. Details regarding product definition and customer needs are incorporated by references to individual product plans (see Fig. 8). Thus, the product portfolio plan provides the link between an organization's platform strategy and underlying business goals. As such, it acts to align and unify both platform and development teams and to set the context for individuals in the organization who are charged with working out the implementation details for the platform strategy.

Fig. 7. Platform version of the software lineage chart given in Fig. 6.



Within HP, the portfolio plan tends to be a collection of slides built around an annotated product lineage chart. The product lineage chart is modified to show the flow of software from the platform to individual products and includes a development time line. This package of materials is augmented with details on the goals and objectives for the platform and product teams. The loose structure and informal nature of HP's portfolio plans work well as long as there is constant communication to reinforce the underlying message about the organization's chosen development paradigm and the link between this paradigm and the organization's business goals.

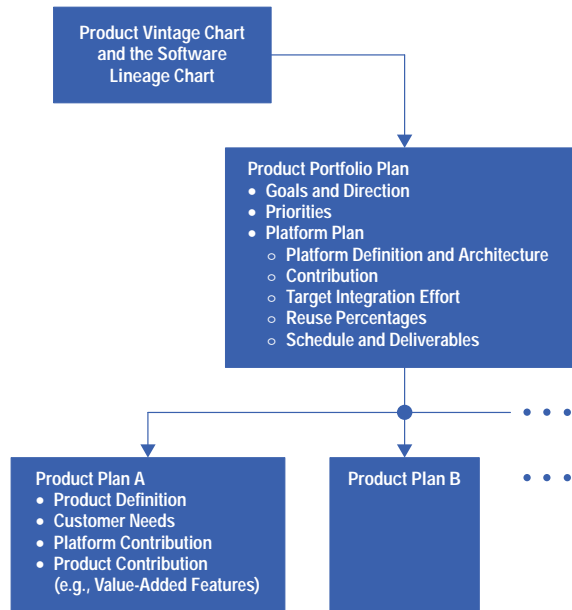
It is essential that all players understand why an organization has chosen the platform development paradigm and what it hopes to achieve as a result of this choice. HP's experience reveals that the rationale and expected benefits leading an organization to adopt the platform development paradigm must constantly be reaffirmed by management. HP divisions that have aligned their development labs around the platform strategy, engendered confidence throughout their organizations, and provided ongoing direction and support during the transition to the new ways of working have adopted platform development more rapidly and achieved better success than divisions lacking active management sponsorship.

Architecture Definition and Partitioning

Since platform development involves separating out the common elements contained within a product family, having a clear and explicit platform architecture becomes very important. This is a key shift compared to traditional product development in which a product's architecture is often implicit and may not even be written down. In fact, often the architectural knowledge of a product or set of products is contained within the head of a single architect or small group of architects. Implicit and informal architectures work for serial independent projects and even for parallel product development when team sizes are small, the level of complexity is low, and the amount of concurrent development is minimal (see Figs. 2 and 3).

These conditions make it possible for the architect or architects to explain the product architecture informally and to assist other engineers as they develop their code. However, increasing complexity and simultaneous pull from multiple teams

Fig. 8. The components of a portfolio plan.



often forces architects to spend most of their time communicating and helping others with the product architecture. While this would be an extreme in the case of independent products, it is virtually certain to happen in the case of a platform. In addition to supporting others, architects need to have time to focus on evolving and extending the architecture.

The solution to this situation is to have architects document and make explicit the platform and product architecture. Formal architecture documents (diagrams and text) make it possible for engineers to access the architectural knowledge that they need to complete their design and implementation tasks without having to refer to the architects constantly. A documented architecture also provides a means through which development teams can provide feedback to the architects so that they can tune and evolve the platform and product architecture. This not only directly benefits the architects, but also helps to ensure that a set of high-quality and better-integrated products result. Having an explicit architecture also makes it possible to quantify trade-offs between the platform and the product in a systematic way and feeds the management planning processes for current and future products.

Another key distinction of the platform architecture is that it subsumes the partitioning between the platform and platform-based products. In this respect it differs from traditional product architectures which usually do not differentiate between individual products within a product family. Like all architectures, the platform architecture typically includes major functional or technology subsystems and the interfaces between them. The chosen partitioning of architectural responsibility between platform and product teams determines the degrees of freedom that product teams have in their work. The shared challenge for platform and product architects is to determine where to draw the platform and product boundary. The boundary needs to be drawn so that it balances the foundation and the infrastructure provided by the platform with the amount of flexibility needed to support value-added product features.

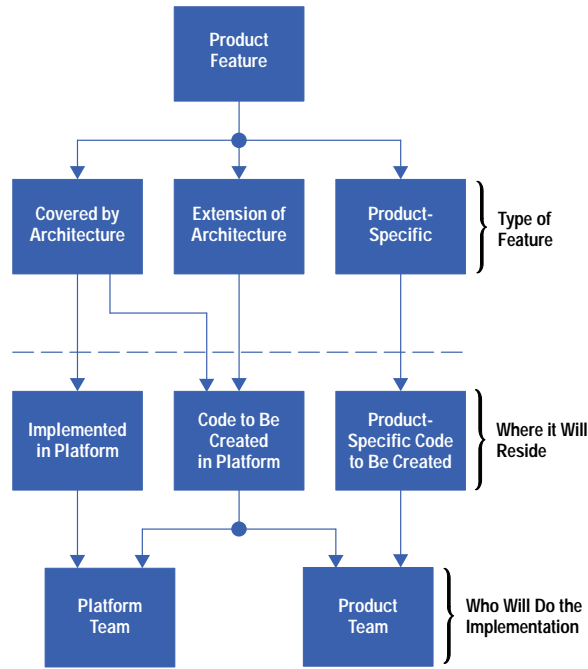
HP organizations struggle in their selection of the initial boundary. In practice their initial partitioning is adjusted over time to achieve the proper balance between platform contribution and product flexibility. Another lesson learned is the need to delineate clearly which interfaces are jointly owned by platform and product teams and which are separately owned. Translating architectural constructs down to the level of interface, module, and code ownership helps avoid conflicting and uncoordinated changes to the platform. It also makes it easier to trade off changes explicitly since it provides a means of linking to all directly impacted teams.

Product Feature Mapping

A closely related area of the platform development model is product feature mapping. This involves the translation of customer needs into product features and ultimately into code implementation. The process of going from customer needs to product feature definition is unchanged from traditional methods. A key shift occurs in the step where product engineers figure out how to map their features to the architecture. Product engineers may no longer have full control in cases where this mapping logically places part of a feature within the product and another part within the platform.

Fig. 9 illustrates the decision-making process that engineers go through to map their features to the platform and product architecture. The ultimate goal of this process is to get engineers to understand what they need to implement and where it falls within the code base. The productivity of product engineers is largely determined by how well they can apply the architecture and move on to implementation. In the case of one HP division with a newly created object-oriented architecture, product engineers were not able to use the architecture at first. The engineers' lack of experience with

Fig. 9. The processes involved in mapping product features to platform and product architectures.



object-oriented concepts and the fact that the architecture was separated into logical, functional layers that did not directly correspond to product features made it extremely difficult for them to understand how to use the architecture. As a result, no product development progress was made.

The underlying lesson learned from this experience was the need for the platform architecture to be explained from a product feature perspective, the way in which the product engineers thought about it. This division's dilemma was solved when the platform architect made explicit the process for mapping features to the architecture and taught this to the product teams. What the architect and the rest of the platform team did was to walk through the process and provide direct coaching to help product engineers work through mapping their features to the platform architecture.

An equally critical dimension of product feature mapping is the understanding by product project managers. Since project managers are responsible for the schedule and resource plan for the product, they need to know what work their team will do and what specifically will be provided by the platform. Consequently, the managers must understand the partitioning of work across the platform-product boundary and also the overhead costs of incorporating reusable components from the platform. This information in turn enables them to create an appropriate work breakdown and arrive at a schedule for their project. The underlying shift here is to a new way of product planning. Within HP we have found it extremely beneficial to provide training and coaching to product project managers to help them with their planning tasks.

Test Architecture and Strategy

The platform architecture not only helps define the dividing line between the platform and product, but also drives changes in test strategies and implementation. Since the platform provides components, modules, and subsystems to product teams before final system integration, some degree of testing of platform work products is necessary before they are delivered to the product teams. Ideally, the platform team fully tests its work products so that product teams can focus on their specific extensions to the platform. This division of test effort results in an overall reduction of the effort, defects, and schedule once the first platform-based product is released.

Unfortunately, several factors complicate this ideal. It may be difficult to fully test platform functionality without additional product-specific code, and in many cases platform functionality is developed concurrently with product functionality rather than sequentially. Both result in lower code quality and potentially increase the testing burden placed on product teams. Product teams usually cannot afford to be the de facto systems test organization for the platform because doing so compromises their own goals, and in particular, their schedule. If product teams get overwhelmed with defects or integration problems passed on by the platform team, conflicts in schedule, priorities, and even team relationships arise. Furthermore, since platform and product teams often sit in close proximity, problem solving gets driven by personal priorities and urgency rather than an objective, organized approach. As a result the organization's cumulative testing effort may actually increase, negating any potential savings.

Thus, a key factor of successfully using the platform development approach is the creation of a shared test architecture and strategy that ensures the delivery of high-quality platform components, thereby enabling product teams to focus their development and testing efforts on product-specific features. The goal of the test strategy is not to outline exhaustively the

details of how the appropriate level of quality is built into deliverables, but rather to describe the overall risk management and test approach. The test strategy makes reference to specific product milestones, checkpoints, and activities. As expected, code drops correspond to key points of synchronization between platform and product teams. At each code drop, there are specific outcomes, questions, and measures that describe both product and platform goals. Based upon these expectations, testing and risk management activities can be determined. The test strategy specifies what these activities are and when they occur, but not the details about their execution. For instance, the test strategy may call for design reviews and inspections at particular points along the life cycle. The specific kinds of reviews or inspections, to what degree, and of what documents, will depend on the types of risks that need to be mitigated.

The test architecture focuses and streamlines the multilayer, multicycle test process. Each element in the test architecture is linked to the product architecture at the component, subsystem, interface (integration), system, or solution level. The individual elements also serve to set the scope and purpose for test suites. For instance, multiple suites may exist to test subsystems for functionality, usability, performance, or reliability. Once each test architecture element is defined, it gets mapped to the test strategy and reconciled with development and milestone dependencies.

Without a test architecture to define the scope and purpose of a test suite, cycles in the test process will often be redundant, increasing the time and resources used in each product team. A product and test architecture can also facilitate the development of a platform regression test strategy. As a platform is used in more projects, the platform team will want a method of ensuring that changes made to the platform don't inadvertently impact the functionality of one product over another.

Organizational Structure and Work Partitioning

In addition to the many architectural implications for developing products under the platform development paradigm, there are many management issues that need to be addressed. The spectrum of management concerns is quite broad and includes defining the context within which teams are configured, deliverables specified, and conflicts resolved, and defining how teams communicate with one another. One pivotal management activity is the definition of the organization's structure and its processes for partitioning work.

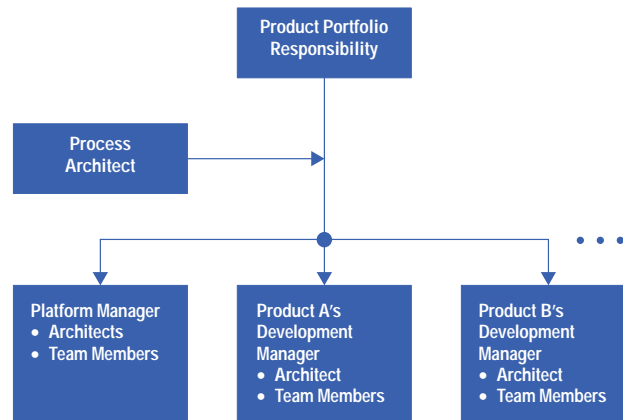
Crucial differences in individual roles between platform development and traditional development paradigms are at the heart of structural and work-assignment issues. Platform development does not result in the creation of new roles within an organization, rather it causes existing roles to become explicit and more formal. In traditional development projects, one or more individuals fulfill the roles of program management, project management, people management, product architecture, and process architecture. A brief definition of each of these roles is contained in Table I.

Role	Responsibilities
Program Management	Integrate and coordinate all the functions involved in developing the product, including marketing, development, learning products, and field support.
Project Management	Develop and maintain the project budget and schedule, allocate resources, and manage work assignments.
People Management	Perform employee training, skill development, performance evaluation, salary administration, and other administrative and legal tasks.
Product Architecture	Develop and evolve the product architecture including coaching and mentoring others on its application and use.
Process Architecture	Define and integrate the various processes used during product development, including how work is done, extensive communication, and process training.

Until fairly recently within HP, the roles of project management, people management, product architecture, and process architecture were filled by a single project manager. This is still common in some HP divisions. As a result of increasing product complexity, many divisions have created explicit positions for product architects and have thereby removed this responsibility from their project managers. Some divisions have gone farther and pulled process architecture responsibilities from their project manager positions and added new process architect positions. The separation of responsibilities is especially important for platform development since multiple teams are working in parallel on products within a single family. The larger scale of this endeavor makes it difficult for any one individual to juggle combinations of these roles while simultaneously attending to the broad scope that accompanies each role. As a result, individual jobs tend to be more specialized and better defined in this paradigm.

The key shift in organizational structure for platform development occurs when separate platform and product teams are created (see Fig. 10). The separation of platform and product teams becomes a necessity when there is more than one

Fig. 10. The organizational structure for the platform development paradigm.



product under development at any given time. HP's experience has shown that having one team simultaneously develop a platform and a product while another team works on a different product is unworkable because of a perceived lack of trust between teams. The underlying issue here is perceived favoritism by the platform manager for the platform team's own product effort. This perception is hard to counter and requires an explicit way to address conflict of interest issues. Our solution is to separate the product responsibility from the platform to ensure that the platform effort is equally shared between the various products. This solution takes advantage of an organization's reporting structure and relationships.

Just picking an appropriate organizational structure is not sufficient for ensuring that work gets done smoothly. Individual work assignments need to be aligned with the reporting structure. Otherwise, individuals within the organization can wind up spending a lot of their time trying to figure out who does what for whom. In HP's experience, formalizing individual roles and responsibilities helps a lot. It simplifies the tracking of individual accountability and provides the context for optimization of work assignments. In addition, management leadership and direction are needed to help people cope with ambiguity and to address coordination of activities across team boundaries.

Partnership Model and Contract

Management, through its own style and working relations, sets the tone and context for how teams work together. As a result, management behavior largely determines the kind of partnerships that will exist within the organization. The purpose of having a partnership model and contract is to make explicit how teams are to work together.

The reuse of software components fundamentally involves a producer-consumer relationship in which one or more teams produce software assets that other teams use. In the case of platform development the platform team is the producer and the product teams are the consumers. How the teams work together determines the overall success of their combined effort. Team perceptions of autonomy, accountability, and control all weigh heavily in setting the context and boundaries for how teams can work together.

The starting point for establishing a solid working relationship is given by the organization's existing social and cultural norms. Within HP we have found that if either team seeks to gain sole control of the relationship, the platform development system will fail. The key shift that is needed is for all teams to think and act as full partners. The notion of working together on a collaborative effort creates a win-win situation and avoids the inherent conflict in one team being superior to another. It also blurs the distinction between platform and product team roles and thus provides a greater degree of flexibility in work assignments (refer back to Fig. 9).

While the partnership model defines the preferred working mode between teams, the partnership contract is the ultimate agreement that is reached between product and platform teams with respect to their mutual deliverables. As such, the partnership contract incorporates and builds upon the partnership model. It contains the specifics of what gets delivered, by whom it is delivered, and when it is delivered. It also includes the explicit communication channels that will be used between teams as well as the mechanisms and protocols for handling changes and exceptions. The partnership contract is not a legally binding document, rather it results from discussions between platform and product teams.

Within HP the process of negotiating a partnership contract is more important than the contract itself. For each new product, the corresponding platform and product team members sit down and work through a series of questions to arrive at a mutual agreement that supports the organization's product portfolio plan. The final answers to the questions can then be incorporated into the platform and product team plans as appropriate (see Fig. 8). This negotiation process serves as a mechanism for joint planning and lays the foundation for the ongoing relationship between the platform and product teams.

Management Processes and Steering Teams

Management processes provide mechanisms for managing ongoing team relationships and for addressing changes in both internal and external requirements. The processes include mechanisms for interproject planning and resource prioritization, tracking and controlling progress, and recognizing, escalating, and resolving issues. At an aggregate level these processes need to be aligned with the product portfolio plan. Management is responsible for achieving alignment and for providing their people with the means to work toward the overall strategy.

A key shift in platform development is the creation of standing teams to deal with ongoing issues. In particular, management typically charters a management steering team and an architecture steering team. The management steering team is made up of the portfolio manager, the platform manager, the product team managers, and the process architect. The team may also include the manager of a separate quality or testing team. The team is responsible for monitoring progress, maintaining resources and schedule synchronization, and resolving daily operational issues. The existence of the team is not meant to replace ongoing, one-on-one work. Rather, team meetings serve as a forum for surfacing issues and establishing linkages for issue resolution.

Architecture steering teams are staffed by senior designers, architects, and technical people. They also include an organization's process architects. Their charter is to focus on managing the overlap between the platform and product architecture. The team is responsible for ensuring that the architecture can be used by product teams effectively and for managing the evolution of the architecture so that overall architectural integrity is preserved.

Within HP, management and architectural steering teams are used at multiple levels. The number of teams and their structure depends on the complexity of a division's business, the number of product lines, and the number of distinct platforms within those product lines. Generally, one steering team is created per platform. In our experience steering teams work well when they are staffed with key stakeholders and have clear, well-articulated charters. Management needs to set appropriate team expectations and model new desired team behaviors, especially if they differ from the organization's existing norms.

Communication and Feedback Model

The success of an organization's platform development system is largely a function of the strength of its communication and feedback paths. Good communication between platform and product teams is essential for reducing unexpected surprises and supporting rapid decision making. For communication to be effective, the right information must reach appropriate individuals in the organization at the proper time. Incorrect, inappropriate, or out-of-date information has little value, and in fact, can be counterproductive.

The attributes of a good communication and feedback model are that it:

- Specifies communication roles and responsibilities
- Enumerates the taxonomy of information types
- Identifies explicit communication links, channels, and pathways between teams
- Establishes triggers for certain types of informational exchanges
- Creates a context for continual organizational learning.

The communication and feedback model can be thought of as an architecture for the movement of information within an organization. As such it plays a major role in helping to ensure that the right information gets to the right place at the right time.

The key shift for most organizations is coping with the need for wider dissemination of information. As the number of interdependent teams increases, the number of stakeholders with interest in a given piece of information increases. Putting together a communication model in the form of a data flow diagram helps teams identify who needs to know about plans, assignments, issues, status, best practices, and successes. However, getting information to flow is not enough because the recipients of the information need to be able to respond and act on the information, if appropriate. The challenge for individuals transmitting information is to gather feedback on the effectiveness of their communication and to tune future information exchanges. Having individuals automatically check their communication effectiveness serves to build and promote organizational learning.

Within HP we have seen significant gains in organizational performance as a result of eliminating communication slippage and optimizing its efficiency. Pleasant, clear communication lowers organizational stress and makes it easier for people to work together. It also enables faster and higher-quality decision making. Many divisions are using e-mail and World-Wide Web publishing to make platform information more readily available to product teams.

Support Model

It goes without saying that for product teams to be effective, they must be able to understand, incorporate, and successfully use the platform. This includes the platform architecture, its components, development tools, and the underlying process infrastructure. The support model addresses how product teams get assistance as they work to incorporate platform components into their products. It provides the means by which product teams get help in the following situations:

- Achieving understanding and resolving “how to” questions (e.g., How do you do X? How does Y work?)
- Sorting out instances of something not working as expected or not meeting a product’s needs (e.g., There appears to be a bug in X. Can Y be adapted to support product feature Z?)

The creation of a support model starts with the identification of detailed support requirements and establishes the mechanics and logistics for how platform and product teams work together. It includes both initial training and ongoing support during implementation. The support model identifies the types of support provided, the mechanisms through which it is delivered, and the overall service level expectations (e.g., typical turnaround or response time). The model covers activities such as providing documentation, delivering training, answering questions, making defect repairs, and releasing enhancements. It also sets forth support roles and responsibilities—to whom to go for assistance. Finally, it includes an escalation path for resolving impasses.

The concept of a support model is not new. In fact, most organizations have an explicit model for supporting their customers. The key shift in platform development is the creation of an explicit support model for in-house product development work. The need for putting formal structures in place increases with the number of product teams working in parallel. In HP’s experience the most effective platform support models are developed jointly by platform and product teams. We have found that including a set of agreed-upon performance measures serves to calibrate and set individual expectations. Typical measures include support availability, response time, and limits on the maximum number of hand-offs. Finally, the support model’s scope extends beyond product construction and needs to include product planning and testing.

Platform and Product Life Cycles

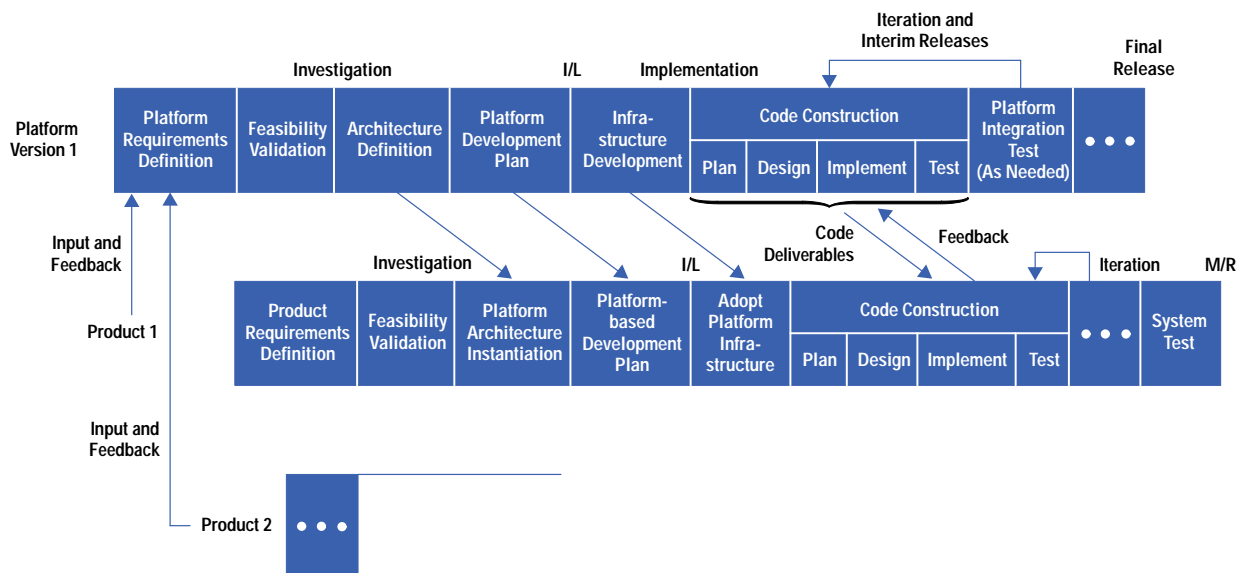
In addition to the need for more formalized support, platform development results in many changes to an organization’s development practices. The development process changes are incremental in nature and generally reflect a formalization and refinement of existing practices. An organization’s platform and product life cycles provide the structure and context within which individual processes fit. Like other life cycles, they are characterized by major phases with associated goals, activities, deliverables, and checkpoints.

Fig. 11 shows the underlying structures of the platform and product life cycles. The major phases of the two life cycles are very similar. Since the platform architecture and components flow from the platform into products, there is a dependency between the two life cycles. The close coupling of the two life cycles represents a key shift compared to autonomous development projects.

The output of the platform investigation phase is a definition of the overall system architecture and answers to the following questions:

- What is the platform?
- How is it intended to be used?
- How will it be delivered to product teams?
- How will it be supported?
- How will it be extended and evolved over time?

Fig. 11. The platform and product development life cycles.



As part of this phase, the platform team identifies necessary changes to existing development practices needed to support platform development. The proposed ways of doing things form the basis for what is called the *platform way*. The deliverables from the investigation phase of the platform need to be essentially complete when product teams move from requirements definition and feasibility validation to instantiating the platform architecture and building upon the platform plan. There is a similar dependency in the implementation phase between infrastructure development and use. The infrastructure consists of the processes and tools that support the platform way.

The implementation phase of the platform life cycle often overlaps with product implementation work. This arrangement requires coordination between iterations of platform and product code construction. The use of an evolutionary development methodology, which subdivides code construction into a series of short plan, design, implementation, and test cycles, provides one way to achieve the needed coordination (see [Article 3](#)). In addition to continuous feedback and integration of platform and product components during implementation, continued support for platform use is essential. As mentioned earlier, the test architecture and strategy will determine the overall approach to verification and validation by the platform and product teams.

The platform life cycle does not end when the platform components for all currently active products are finished. Rather it wraps around and a new investigation phase begins. In subsequent iterations the platform architecture, processes, and components are modified and extended based on new requirements. The boundary between the platform and products may change as certain features migrate into the platform so that they can be used by subsequent products.

HP's cumulative experience underscores the need to separate platform and product life cycles. Furthermore, life cycles must be linked to existing hardware life cycles, or software work may not begin until after the hardware prototype is done, thereby delaying product introduction. We have also learned that securing future product input during the development of the overall system architecture and definition of the platform way is essential to building strong interteam relationships. One particularly effective way to engage future product teams is to involve them in signing off on platform checkpoints.

Development Model and Development Process

The development model and the development process describe how new functionality is created. They cover the processes used in the creation and enhancement of a platform module through its integration into products. They are analogous to a module or component life cycle in that they catalog the development steps spanning from construction to final use. Since platform modules and components are ultimately used by product teams, the development process really includes two different perspectives: software asset development and software asset utilization. The key shift in platform development is formalization of these perspectives.

In cases where the boundary between platform and products is diffuse, a single process incorporating both perspectives can be used. The common process can be applied by both teams regardless of whether they were building platform or product functionality. On the other hand, a sharp boundary has the advantage of providing product and platform teams with greater autonomy over their work, since each can use different processes. For example, if the platform-product interface is confined to linking a set of library modules, then the platform might be designed and built using object-oriented methods and tools, while product teams might use traditional structured programming methods and tools. This decoupling is not without cost because the use of different methods and approaches makes it harder for teams to communicate—consider the difference between object-oriented programming in C++ and functional programming in C. Furthermore, its very existence raises the cost of modifying the platform and product boundary and makes it significantly more expensive to migrate functionality across the boundary. It also reduces resource flexibility by making it more difficult to move engineers between the platform and product teams.

Experience within HP has shown that platform development proceeds smoothly when platform and product teams follow highly complementary development processes. By agreeing to use common methods and tools, platform and product teams make it easier for one another to cross the boundary between their work domains. This provides flexibility so that resource use can be optimized. It also allows for evolution of the platform through incremental redefinition of the platform-product boundary. HP division's have reaped significant benefits from having documented development processes since these reduce the support burden for bringing new engineers up to speed on how things are done.

Delivery Model

The delivery model defines how platform modules, components, and subsystems are passed on to product teams. Platform deliveries are essentially a microcosm of the product release process since they cover the depth and breadth of what a development organization delivers at the manufacturing release (M/R) of a product. At M/R, a final production build is delivered to manufacturing along with a set of release notes, documentation, and other supporting material.

Final release may be preceded by multiple iterations and intermediate releases, especially in the case of complex systems products. Furthermore, each constituent part of the whole product can be delivered in a different way. For example, while physical hardware and firmware go to manufacturing along with detailed assembly, calibration, and packaging instructions, the product's software drivers may simply be given to a third party for replication.

Although most organizations have long had internal hand-offs among teams, the key shift in platform development is making these handoffs and deliveries explicit. While the platform life cycle outlines the types of artifacts that get delivered and

when they are delivered, the platform delivery model provides specifics on what is delivered and how it is delivered. If a delivery is analogous to passing a package from one team to another, then the delivery model corresponds to a packing list or bill of materials.

The delivery model also specifies how the package is delivered to the product teams. The delivery mechanism typically falls somewhere along the continuum from push to pull. In the push case, the platform team delivers packages when they are ready. In the pull case, product teams request packages when they want them. There are pros and cons to each approach. The push approach can force product teams to use something before they are ready, while the pull approach can cause the platform team to support multiple versions of the same component. Within HP most divisions have adopted a hybrid model in which both push and pull approaches are used depending on the type of each deliverable.

Although it might be tempting to rely on an ad hoc delivery model, HP's experience shows that having an explicit, formal delivery model is essential to ensure that product teams can successfully use the platform. This is particularly true when multiple product teams are simultaneously using the platform. Furthermore, the delivery model provides additional means of supporting product team members.

Validation and Test Processes

Another important aspect of platform and product development is the accompanying validation and test processes. These processes correspond to the tactics used to implement the overall system test strategy and architecture, including the selection, implementation, and execution of appropriate testing methods and procedures.

In traditional product development, testing is often relegated to the back end of the life cycle and almost becomes a certification step intended to ensure that the product meets certain quality and reliability requirements. In contrast, progressive development organizations view testing as a process of verification and validation that can be applied throughout the product life cycle. These organizations conscientiously perform verification and validation activities as early as possible in their life cycles to catch defects early and thus reduce overall development effort and shorten back-end cycle time.

Organizations that have successfully made the shift to early defect detection see testing for platform development as a variant of their current processes. For others, the move to platform development forces a shift in testing emphasis to architected, early defect detection. When this shift is not made in time, platform development projects become very painful and can even fail.

The key shift needed is for the platform team to certify its work products before they are delivered to product teams. Many testing techniques can be used to ensure appropriate levels of quality throughout the development life cycle. Techniques such as formal design techniques, reviews, and inspections can be used to catch errors before moving into implementation. Prototypes can be used to demonstrate feasibility and validate certain design constructs. Coding standards help to ensure code portability and bounds checking is accomplished via assertions in the code. Downstream activities include white box and black box testing, unit testing, and regression testing. The stability of code modules or product components is verified by doing frequent, regular builds. Finally, integration testing is used to ensure proper cohesion between platform and product components. This list of possible verification methods is by no means exhaustive and many additional types of tests and quality methods exist. It is incumbent on the teams to pick those methods that best address their particular project risks.

Once the platform team has developed processes that meet the delivery criteria of the product teams, the second key shift is for the product teams to leverage and complement the validation and verification already accomplished by the platform team. This shift involves making changes in the product test process to leverage the test architecture and focus test cycles on product-specific contributions and their integration with the platform. If product teams find themselves testing platform components, then something is wrong. This is analogous to having teams that use C compilers do testing on the accompanying C libraries.

Experience within HP indicates that when the product test strategy is not revised, unnecessary redundant testing occurs. In this case the platform is fully tested as part of every product. As the number of simultaneous products under development increases, so does the burden placed on those doing system testing. The end result is repeated testing of parts of the platform, at high cost, and with little added risk reduction.

Development Tools and Infrastructure

The final element of how engineers undertake their work is made up of the development tools and infrastructure that they work with. The tools needed for platform development are no different from those used in other development activities. However, platform development often demands more from the development tools and infrastructure than traditional product development. Since platform development is usually accompanied by a higher degree of complexity, tool flexibility and robustness often become issues. Key shifts in this area have to do with (1) formal tracking of project issues, (2) architecture, design, and process documentation, and (3) robust configuration management.

Ad hoc issue management breaks down in the case of platform development because of the large number of stakeholders external to the platform team. The use of a process and tool to maintain a database of issues and their resolutions not only

supports the ongoing issue management, but also provides a means of capturing historical information about key project decisions.

Design documentation and automation tools help with the generation of documentation that is essential for successfully supporting product teams. Standard design tools also help to ensure consistency across the work of the members of the platform team by providing common formats for individual work products.

Finally, tools for simple version control are replaced by a full-fledged source configuration management system that provides the needed horsepower to address concurrent platform and product development needs. In HP's experience moving to a robust source configuration management system leads to new ways of working. The transition to more sophisticated processes for version, build, and workspace management is nontrivial.

Metrics and Measurement Processes

Metrics and measurement processes cut across all the other elements of the platform development model. These processes are used to know how things are going and to flag potential exceptions. Although any organization can collect metrics, the real question is whether or not the organization's metrics are driving effective decision making. A good metrics program measures the right things and then provides a means for acting on the data collected.

Since metrics communicate what is important and focus organizational energy, they provide a mechanism for shaping and institutionalizing the platform way of doing things. Consequently, they provide a means of reinforcing the new behaviors that are required for platform development to be successful. A set of metrics can be defined for the elements of the platform model in such a way that the metrics correspond to how well each element is working for the organization. Defining this set of metrics, rolling them out, and following through correspond to the key shifts in this area.

The management processes and steering teams and the platform and product life cycle directly link many of the metrics to decision making within an organization. An organization's management processes and steering teams provide the context for ongoing review of metrics and for taking appropriate follow-up actions, particularly those related to planning, management, and support issues. With respect to the architecture and development elements of the platform competency model, metrics are included as part of the platform and product life cycles. The metrics for individual elements are tied to specific life cycle checkpoints (refer back to Fig. 10).

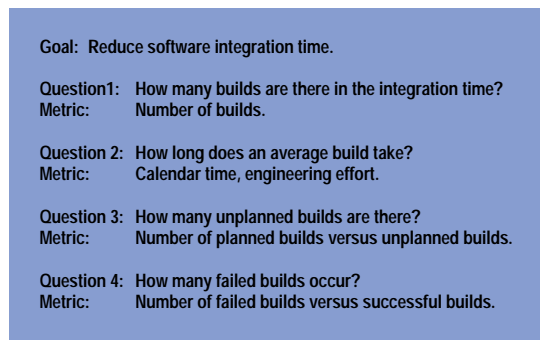
The metrics that an organization collects and uses largely depend on its business requirements and level of maturity. A technique that has been particularly effective within HP is the goal/question/metric paradigm (GQM).² The principles behind GQM are rather simple. It begins with a goal or set of goals, defines a set of questions that provide visibility into how well the organization is doing at meeting the goals, and identifies a set of measures that provide answers to the questions (see Fig. 12). Since GQM is not necessarily linear, it can be used as a means of exploring and clarifying the meaning of a stated goal. This makes it a powerful technique for defining the metrics for the individual elements of the platform model.

Values and Reward System

In most organizations individuals act based on their beliefs, values, and understanding of the consequences of their actions. Although people do extraordinary things when presented with dire consequences, in most cases an extraordinary level of performance is not sustainable. High turnover and burnout in development organizations are often the result of continually demanding individual and team heroics. The norms for individual and team behaviors depend on an organization's culture which in turn is shaped by the collective beliefs and values of individuals within the organization.

The importance of shaping organizational culture to support platform development cannot be understated. The first step in making the transition to new behaviors for platform development is to identify the vision for how things will be working when platform development is fully functioning within the organization. The vision needs to be vivid, rich, and descriptive. Having a vision is not sufficient to get an organization to its desired state. The organization's culture, values, and rewards must be aligned with the vision.

Fig. 12. An example of the goal/question/metric paradigm (GQM).



Goal:	Reduce software integration time.
Question 1:	How many builds are there in the integration time?
Metric:	Number of builds.
Question 2:	How long does an average build take?
Metric:	Calendar time, engineering effort.
Question 3:	How many unplanned builds are there?
Metric:	Number of planned builds versus unplanned builds.
Question 4:	How many failed builds occur?
Metric:	Number of failed builds versus successful builds.

The organization's recognition and rewards systems play a major role in reinforcing the desired new behaviors that are part of the platform way. It is extremely important to align performance evaluation, recognition, and reward mechanisms with behaviors that simultaneously support achieving platform, product, and business goals. A common failure is to assume that all of these mechanisms and systems are aligned. The key shift is to link the desired state explicitly to specific roles and activities within the organization and to make sure that those roles, activities, and behaviors are reinforced.

Within HP we achieve alignment by working issues both top-down and bottom-up. As a result we integrate business thinking with the logistical, operational, and personal attributes of vision in such a way that individuals throughout the organization know how they fit in. We have also found it necessary to revise existing performance evaluation criteria to tie them to an organization's platform development strategy.

Results

Various divisions within HP have realized productivity and efficiency gains as a result of adopting the platform development paradigm. The degree of improvement varies between divisions and further gains are expected since many divisions are still in the midst of their transition.

One division has doubled its product generation capability from an average of two to four new products per year with no increase in development staff. Another division has cut its time between product releases (TBSP) from twelve to six months and has slowed staffing growth despite an exponential increase in the number of new product releases per year. This division's staff grew linearly while the number of product releases went up almost tenfold over a three-year period. The experience at another division included increased product consistency, improved similarity between products within a product family, and better overall quality. These benefits ultimately enabled this division to offer a better integrated solution to its customers. Furthermore, this division cut its time to market for new products and has reduced the time required to bring new staff on board and make them fully productive.

Fortunately, organizations do not have to wait until they complete the transition to platform development before they start reaping benefits. Since platform development mandates a higher level of organizational maturity and individual skill, it helps to institutionalize solid engineering practices. Typically, there is an incremental productivity gain associated with each improvement in existing engineering practices. Thus, an organization's transition to platform development effectively compounds multiple productivity improvements. Benefits can be reaped all along the journey and not just at the end. In fact, for those HP divisions that have had to cancel or delay their platform efforts, most are better off than before their migration to the platform development paradigm. The reason for this is quite simple: they continue to use better engineering practices on new and existing projects.

In comparison to the business impact of platform development, the results from using the platform competency model in development are not quite as dramatic. The model has primarily been used as a tool for exploring the many facets of platform development. It has proved to be an excellent vehicle for building a shared understanding of what it will take to institutionalize platform development within an organization. The model has also been used as a diagnostic tool to identify areas requiring further attention, thus providing the basis for developing an overall investment plan.

Significant differences exist between the investment plans of different organizations. Differences in the magnitude and sequencing of investments in the elements of the competency model can be traced to differences in an organization's respective level of software expertise, the maturity of its operating practices, and its business constraints. Since each organization has a different profile, its roadmap for establishing a platform capability is uniquely its own. In our experience, when organizations stumble in their adoption of the platform development paradigm, the root cause of their difficulty can be traced to one of the model's elements.

The feedback that we have received as a result of using the competency model with HP divisions has only served to validate the model. There have been a few minor additions to the model, but no significant changes. The unanimous consensus within HP is that each and every element of the model is critical. All sixteen elements of the model are necessary for successful deployment of platform development.

Conclusion

Although numerous product development strategies and paradigms are in use throughout HP, platform development is becoming the paradigm of choice within HP. The platform competency model discussed in this article captures the essence of HP's cumulative experience in platform development. The model is being used by HP product development organizations to understand the requirements and implications of platform development, to guide the creation of investment plans, and to assist in the customization and tailoring of the model's elements to fit each organization's particular situation.

Engineering Perspective

From an engineering and technical perspective, the platform development paradigm is an enabler for technologically superior products. The use of a platform as the base for new products allows engineers and architects to focus their efforts on the key, new technical contributions. The use of a platform also results in more solid products that are higher in quality, better integrated, and more consistent. This degree of robustness results from the continual evolution and improvement of the platform.

Management Perspective

The platform development paradigm results in a number of changes in the way development teams are managed. Since new products are developed using both platform and product component streams, management attention shifts from an intense product focus to a more integrated and global perspective. Managers find themselves actively involved in charting their organization's future, in setting appropriate goals and objectives, and in establishing the necessary supporting infrastructure. Managers through their own actions shape how their organization transitions into new ways of working within the platform development paradigm.

In platform development, managers are principal actors in understanding the trade-offs between product and platform team needs, communicating and making those trade-offs explicit, linking them to short-term and long-term project and business goals, and finally, taking appropriate action. Ultimate success is rooted in the ability of the entire management team to align its thinking and actions around its instantiation of the platform development paradigm. Active participation by multiple levels of management, appropriate acceptance, and support are necessary but not sufficient conditions for successful implementation of the platform development paradigm.

Platform development enables an organization to deliver innovative, feature-rich, high-quality, and consistent products on short development schedules. It does so through increased reuse, reductions in per-product testing, and ever increasing product quality. Simultaneously, it raises overall engineering efficiency, makes it easy to assimilate new engineers, and improves schedule predictability. These gains ultimately translate into reduced development cycle times and shorter time to market.

Organizational and Business Perspective

From a systemic viewpoint, the platform development paradigm is just one of many product development strategies. Increased development efficiency and cycle-time reduction can be achieved in incremental steps by gradually evolving an organization's development competencies. Transitioning to platform development is nontrivial and is in many ways analogous to reengineering the product development process. So even though changing development paradigms is a business imperative for some HP divisions, the shift away from largely independent and autonomous development projects to collections of interrelated projects is somewhat countercultural. Successful adoption of the platform development paradigm requires aligning the organizational culture, values, and rewards to support new ways of working.

There are numerous approaches to adopting platform development. Within HP most development organizations tend to follow an incremental, evolutionary approach as opposed to a complete, ground-up, or reengineering approach. The former supports ongoing, new product development efforts and gradually improves an organization's development capability. As such it reduces the amount of change the organization must assimilate at any given time and stretches the change investment out over a longer period of time.

Regardless of the approach that an organization selects, there are many common issues and challenges that it will face and yet each organization inevitably faces some unique issues and challenges. How an organization goes about implementing the capabilities underlying the model's individual elements will vary depending on its situation. Each organization needs to work out an implementation plan that fits the parameters of its business context.

Acknowledgments

I would like to thank the many individuals within the software initiative who have been working in partnership with HP divisions to implement platform development. I believe that they are truly making a difference for HP by helping divisions reach a new level of competitiveness in their software development effort. The platform competency model is the result of their hard work and the many learnings from actual applications of the model. I would like to extend special thanks to Todd Cotton, Ron Crough, Mark Davey, Joni Ohta, and Mike Ogush for working with me on the definition, application, and evolution of the platform competency model.

References

1. G. A. Moore, *Crossing the Chasm: Marketing and Selling Technology Products to Mainstream Customers*, Harper Business, 1991.
2. T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley, 1988.
3. R. B. Grady, *Practical Software Metrics for Project Management and Process Improvement*, P T R Prentice-Hall Inc., 1992.

-
-
- ▶ [Go to Article 7](#)
 - ▶ [Go to Table of Contents](#)
 - ▶ [Go to HP Journal Home Page](#)