The Product Flow Model

Gio Wiederhold
Stanford University
14 May 2003

Abstract

We observed a new approach being adopted in the software industry that combines a business model with strong technical support. This Product Flow Model combines aspects of the Product Line model with workflow concepts for virtual enterprises.

The Product Flow Model (PFM) addresses the costly ongoing maintenance of major products being delivered to customers. When such maintenance is provided by internal information technology (IT) operations for software then little overall lifetime cost reduction has been achieved by reduced software acquisition costs, since maintenance consumes 70 to 90% of total lifetime software costs. A software supplier operating according to PFM also provides the ongoing maintenance to the customer. In order service the customer effectively there must be an effective information flow expressing the needs of the customer to the software supplier. We perceive three classes of needs, leading to corrective, adaptive, and perfective maintenance.

Software supplier that operate within the PFM can be assured of substantial continuing income and great customer loyalty. We will elaborate both on the information flow and on technologies needed to be successful in PFM services.

1. Introduction

There is a close relationship between software delivery technology and the settings where the technology is appropriate. At one end of the spectrum is individual software, where an individual has a problem, formulates it, devises a program, runs and tests it, and walks away with the result. At the other extreme are enterprises that outsource all their computing needs to an outside service, for management, development, and operation. This range is from total control to limited, contractual control. Most practical situations fall in between; if corporate competence is embodied in software, not all control should be ceded. Large enterprises develop software in substantial Information Technology (IT) departments, but also depend greatly on purchased software. Maintaining the total software inventory so that the IT department is responsive to the diverse and changing requirements of an enterprise is an even greater challenge, one not always adequately met. Finding the right balance between control, responsiveness to changing needs, and costs is an important management issue in many enterprises.

We focus on a model that is still evolving. We have given this approach the name Product Flow Model (PFM) because it combines aspects of the Product Line Model with workflow concepts for virtual enterprises.

1.1 Product Line Software Production.

The term *Product Line* has been used by Barry Boehm to describe the process in software companies that have a product type specialization [Boehm:99]. Such suppliers exploit their domain-specific knowledge and existing inventory to rapidly create semi-custom products for new customers. An example might be a company producing payroll programs for medium and large enterprises. Such enterprises have needs and structures that cannot be satisfied by shrink-wrapped products, and yet there will be a great deal overlap among their software needs. As Barry Boehm has demonstrated, a modest initial investment, say 20% over the most economical cost of delivering the first instance of a program, will pay off in the delivery of a second or third instance of a similar program.

Reuse is enhanced by having a software architecture that allows effective reuse of modules in the inventory. The architecture defines what types of functions are placed into what modules and what the input and output conventions are for those modules so they will be able to interoperate. With an architecture comes a layering: infrastructure, servers, middleware, mediators, and end-user facilities [WiederholdEa:92]. New products are created through assembly of modules written for prior instances, modules that are modestly modified, but have similar interfaces, and wholly new modules.

The Product Line Model encourages companies that produce custom software to fund a reuse program so that when a subsequent customer needs a similar product materialized corporate knowledge will be in place so that a new product can be generated rapidly, efficiently, and with few errors [Lim:97]. Many of the large system integrators have adopted such an approach, explicitly or implicitly. The contracts they issue to a customer specify that they have the rights to reuse all the software written for the customer, other than customer specific data. The Product Line approach creates products adapted to serve the needs of a specific new customer.

1.2 Workflow

Workflow defines a structured way of doing repetitive business processes [LeymanR:00]. In PFM the objective is to create an improved version of the previous product. A workflow model identifies the sources of information, the processing points where knowledge in the form of human interaction or computer programs transforms the information, and where the results go. Feedback cycles and storage nodes are a common component in workflow models. Traditional software engineering models also include aspects of workflow, but focus on the product, up to delivery. Only after delivery is the software productive, and that is when relevant feedback can be generated. Dealing with feedback requires maintenance [Basili:90].

Successful software products have many versions, long lifetimes, and corresponding high maintenance cost ratios over their lifetime. Lifetimes before complete product (not version) replacement are around 15 years [Wiederhold:95]. Maintenance costs of such enterprise software amount to 60 to 90% of total costs [Pigoski:97]. Military software is at the low end, but that may be because their users don't complain much. The high level of effort needed for maintenance frustrate many IT shops [Swanson:76]. The naïve solutions, namely help purchasers to write better specifications and exhort software

engineers to make the products more reliable, only address the problems at the initial delivery.

2. The Product Flow Process

An enterprise acquiring Product Flow software delegates maintenance as well as initial development to the vendor. We focus now on vendors that will market the same software product to multiple customers. The diversity of needs of those customers means that there must be fair amount of parameterization in the product. Since, for major customers it is rare that any parameterization can cover all needs, there will also be tools for adaptation to local needs. The enterprise IT department has a much less work, but remains a critical link in the product workflow.

The Product Flow Model considers specifically to the process of creation of upgraded versions of the prior product, based on feedback from ongoing use. The upgraded product will be made widely available to both prior and new customers. This distinction drastically changes the business model:

- A Product Line methodology creates custom products rapidly and economically.
- Workflow specifies the paths that information takes in order to provide benefits
- The Product Flow methodology rapidly and economically creates adapts and updates prior products; products intended to serve a broad spectrum of customers.

2.1 Initialization

The initial product in a Product Flow approach may still be produced via any of the well-known approaches as the waterfall model [Boehm:81] or more likely the spiral model [Boehm:88], or a risk-prioritizing technique as the WaterSluice model [Burback:98]. All these models have as the endpoint the delivery of a product to the customer. The software may be the result of a successful Product Line approach, having become sufficiently general that the product is now a candidate for broader distribution. After delivery to a customer the products enter a maintenance phase.

2.2 Continuing Improvement

In the Product Flow Model the delivery to a customer is not the end point. The supplier following a PFM approach focuses as much on retaining existing customers as on making new sales. PFM customers are expected to use the product for a long time and require ongoing adaptation and improvements. We use well-established definitions for the three classes of long-term maintenance [Marciniak:94]:

1. **Bug fixing** or corrective maintenance is of course essential, but not the end all of Product Flow maintenance. In practice, the majority of bug fixing is performed early in the post-delivery cycle – if it is not successfully performed, the product will not be accepted in the market place and hence not have any significant life.
2. **Adaptation** is needed to satisfy external mandated constraints, as an ability to deal with new hardware, operating systems, network, and browser updates that are used in the customers environment. Governmental regulations may also require

adaptation, new taxation rules affect financial programs, new mergers affect business information systems, and new medical technologies affect health care software [BondS:01].

3. **Perfective maintenance** includes performance upgrades, assuring scalability as demands grow, keeping interfaces smooth an consistent with industry developments, and being able to exploit features of interoperating software by other vendors, as databases, webservices, schedulers, and the like. Perfection may be less urgent, but keeps the customer happy and loyal, important here since continuing maintenance fees will be as important for Product Flow software development as new sale licenses.

The last two classes, adaptation and perfection are known to consume respectively 30% to 40% and 45% to 55% of maintenance costs over the long term [LientzS:80]. In PFM these are handled by the software and service supplier. What is the motivation that drives such work?

2.3 Paying for maintenance

We find that PFM suppliers charge basic maintenance service fees on the order of 15% of original license purchase costs. Some fraction of that will be used for routine customer assistance. Higher levels of support, for instance to provide on-site help, are generally available. A significant amount of the basic fee, at least half, flows to the engineering staff that performs the actual software maintenance efforts. Over the long expected lifetime of the software at a customer site, maintenance fees exceed the initial acquisition license fees.

The customer has reason to be happy to pay the maintenance fees. In addition to obtaining reliable software, the enterprise will also receive updates that will assure compatibility with existing requirements, and receive a better product with each release. We find that the software product does become larger, a growth rate suggested is that such software grows by an amount equal to its original size in every major version, say every 18 to 30 months. Intermediate version releases, containing compatible upgrades, also show growth. The growth rate is initially similar to that seen in hardware capacity growth, but is not exponential. The complexity of software and the human resources at the supplier impose a more linear growth model.

2.4 Replacement instead of maintenance

An alternative to maintaining software is replace obsolescing software with newly written software. We often hear that it would be better to rewrite it all. Rewriting may indeed be better when the prior version was a prototype, and contains messy and patched -up code. But as software programs mature, rewriting becomes less of an option. Issues that mitigate against rewriting software are:

- **Assuring full compatibility:** Even when there are well-defined interface standards, it is difficult to assure that replacement software is fully compatible. There are examples where a group within database systems vendor produced a better product, fully compatible with the published standard, and intended to

replace the prior product. Problems that customers encountered with minor incompatibilities caused the company to lose momentum and eventually close.

- **Exponential cost growth with size:** As software increases in size the interactions among its parts increase and the staff required to write such software becomes disproportionally larger. Since perfected versions will be larger, a much larger staff would be needed to create new versions. [Brooks:95].

In practice, rewriting operationally software is becoming less and less feasible. The replacement of software that could not comply with Year 2000 problems may have been the last large-scale replacement efforts.

3. Information flow

In order to perform the maintenance functions several information sources must be exploited. It is crucial that all input types are recognized and work smoothly. Management support at the software company is essential for PFM to work.

Corrections

For corrective maintenance the traditional reporting methods are available to the customer. A specialized team at the supplier will log error reports, eliminate misunderstandings, combine errors that seem to be identical, devise workarounds if feasible, and forward significant problems to the engineering staff. The engineers, since the supplier operates in the PFM that supports continuing improvement, will include the original developers. Fixing bugs now becomes an opportunity to learn, rather than a task to be dreaded. The next release will be improved in well-considered way. Of course, for critical errors patches may still have to be shipped to the customers that encountered the problem.

Adaptations

Adaptive maintenance is required to keep up with external changes that affect software, sometimes fatally, often just awkwardly. Aspects of our changing world that affect software include:

- Changes in the computational system setting: new hardware, new storage devices, new operating systems, updates of software that our product interacts with. If a product is not adapted in a timely way, then new system acquisitions, promising more performance, are disabled. Soon a decision will be made by the customer: switch or relegate the product to an odd corner of the enterprise.
- Changes in communication: again, adaptation to upgraded communication protocols and methods is essential for survival. Failure to communicate well deprives the product of input or renders its output useless.
- Governmental rules: many software tasks are subject to rules mandated by governmental agencies: tax laws, accounting standards, personnel management, privacy protection, etc. Many enterprises operate in multiple countries, so that the aggregate flow of rules is considerable, and requires constant adaptation of products.

Input for required adaptations comes from software suppliers who make products that interface with the suppliers product, from government sources, and from standards body. A company operating in the product flow model will have representatives at many of these external entities. A company with a substantial market share has an advantage: it will receive warnings from vendors of corresponding products, since those are equally at risk of lack of customer acceptance. Even the gorillas in the software business provide advanced information. In any case, a specialized PFM software supplier is much less likely to be caught unaware than a enterprise IT department, and may have enough clout to influence specifications and deadlines.

Perfecting

Perfection is unachievable in this world, and even an attempt at perfection in software would induce excessive delays. The essence of the Product Flow approach is hence the process of perfecting software, not its result. Perfecting makes existing software work better, it upgrades the functionality, but not to the extent that truly new products are created. Perfecting does not require novel insights, or substantial training, and certainly no retraining at the customer. Improved functionalities are natural, they follow the same paradigm, an often makes the work simpler for the customer.

Motivation for perfecting are many, and will differ for various types of product. We list a few.

- Interfaces: there is today no good science that predicts what human interface functions are effective and which ones are awkward. As products are successful with a certain look-and-feel interface choice, those choices become the habitual standard, and all other software gravitates to the some conventions.
- Globalization: when dealing with an international market it is important that languages and methods are comfortable for all participants. It is impossible for technical developers to be aware of all the cultural differences that affect interaction with computers. Responsiveness to local needs is an important aspect when perfecting software.
- Growth: successful enterprises grow: they will have more employees, more products, more customers in more countries, a deeper organizational structure, etc. Limits that existed in the initial design must eventually be removed, without increasing the cost to smaller, newer customers. A failure to keep up will deprive the product of its largest customers.
- Niche requirements: since Product Flow software is intended to serve a broad customer base, certain user types will want certain minor enhancements: when editing documents, lawyers want to be able to have a strikethrough option in their fonts, while mathematicians want to set formulas according to arcane rules. An enterprise that employs both types will want both features in the same product.

At the same time these improvements cannot affect ongoing customer operations. Making them small, incremental and rapid reduces the risk. A quarterly minor version release cycle seems wise. If something goes wrong in an aspect of perfection, the customer can continue to work for another few months with the older version. Bundling too many

changes into major releases, say every two years, induces major risks that are not offset by spectacular new capabilities.

Not all requests by customers can be satisfied, but a constructive interaction can be initiated. That interaction may also provide motivations for developing new products, initiating a new product flow cycle.

Input to make existing products better requires a well-trained sales and marketing force. Whereas in a single delivery situation sales personnel will gloss over the customers desires when they doesn't match what their software provides, in the PFM setting such input provides the driving force for a company to stay ahead of competitors. Serving an existing customer will also bring much realism into the request flow. Still, not all requests can be satisfied, but a constructive interaction can be initiated.

Development.

The continuity and intensity of dealing with product flow software encourages operating in a mode that is akin to that promoted in extreme programming (XP) [Beck:99]. Both approaches require implementors to interact closely, deal with rapid turnaround, and maintain close linkage with the customer. Since the common setting for XP involves writing software for a specific purpose the customer has a representative within the group. In the Product Flow model the interests of multiple customers are represented by marketing personnel. Other features of XP, as pair-wise programming, common ownership of software, integral testing, no overtime, will be useful in PFM development, but are not essential. The omission of external documentation in XP is probably unwise in Product Flow, since there is a level of indirection with respect to the customer, who may need to add their own adaptations into the delivered product, using the tools provided by the supplier.

Tool development is an important aspect of operating in a Product Flow model. Tools are used to assure ongoing compatibility of the products. Many of the tools provide standard infrastructure functionality, as the ability for the software to work on a variety of hardware platforms, work within multiple operating systems, handle a variety of communication protocols, and obtain data and deposit results using a variety of database management products, typically relational database management systems (RDBMSs). They will also contain tables to deal with alternate interfaces, communication protocols, and allow adaptation to foreign languages, their character sets, and presentation choices.

Developers working in a Product Flow setting must be familiar with work that has gone on before, with the tools that are used to manage the development process. New hires must be trained in the development process. They must also be willing to read and understand the code written by their predecessors. Here is where academic education fails. Students in our schools are taught immediately to write programs, with little exposure to programs, written by experts, that they might learn from. Product Flow companies must engage in retraining here, both of recent graduates and programmers that

have developed more independent habits. The need to use the tool sets reinforces the switch to `read before you write'. Focusing on smaller code segments reduces the risk of serious failure.

Conclusion

In the Product Flow Model maintenance becomes a benefit, not a liability. It engages the customers over a long term. A successful product flow software supplier assumes costly functions for the enterprise IT department, without the enterprise losing all control. Functions that define the essential aspect of an enterprise are likely to be developed and maintained in-house, but can interact securely with product flow software that is installed also in-house.

New customers of the old product automatically receive the current, improved product, and are integrated into the ongoing PFM process. The requirements input and development costs for prior upgrades were, at least in part, supported through the ongoing maintenance process. Existing customers are prime targets for new, related products.

The company selling product flow software and maintenance licenses receives a steady income from its licensees, even if new sales diminish. Losing customers is rare. The cost for an enterprise to move useful acquired PFM software back in-house would incur many times the annual license cost, and would only do so in a disastrous situation. Disasters can happen, of course, and no model can assure continuity But in many circumstances the product flow model and its technology provide a valuable path in serving software exploitation, with benefits to the suppliers as well as the enterprise customers.

References:

[Beck:99] Kent Beck: *Extreme Programming Explained: Embrace Change*; Addison-Wesley, 1999.

[Basili:90] Victor Basili: "Viewing Maintenance as Reuse-Oriented Software Development"; *IEEE Software,* Vol.7 No.1, Jan. 1990, pp.19-25.

[Boehm:99] Barry Boehm: Managing Software Productivity and Reuse; *IEEE Computer,* vol 32, No.9, Sept. 1999, pp.111-113.

[Boehm:88] Barry Boehm: "A Spiral Model for Software Development and Enhancement"; *IEEE Computer,* vol. 21 no.5, May 1988, pp.61-72.

[Boehm:81] Barry Boehm: *Software Engineering Economics;* Prentice-Hall, 1981.

[BondS:01] Andy Bond and Jagan Sud: "Service Composition for Enterprise Programming"; *Proc. Working Conference on Complex and Dynamic Systems Architecture,* University of Queensland, Dec. 2001, Brisbane, Australia.

[Brooks:95] Frederick Brooks: *The Mythical Man-Month, Essays in Software Engineering;* Addison-Wesley, 1975, reprinted 1995.

[Burback:98] Ron Burback: The Water Sluice Method for Software Development; Stanford University CSD thesis, 1998.

[LeymanR:00] F. Leymann and D. Roller: *Production Workflow: Concepts and Techniques*; Prentice-Hall, 2000.

[LientzS:80] B.P. Lientz and E.B. Swanson: *Software Maintenance Management;* Addison-Wesley, 1980..

[Lim:98] Wayne C. Lim: *The Economics of Software Reuse;* Prentice-hall, 1998.

[Marciniak:94] Marcianak *Encyclopedia of Software Engineering*, Wiley, 1994.

[Pigoski:97] Thomas M. Pigoski: *Practical Software Maintenance - Best Practices for Managing Your Software Investment* , IEEE Computer Society Press, 1997.

[Swanson:76] E.B. Swanson: "The Dimension of Maintenance"; *Proc.2nd Int.Conf.on Software Engineering*, 1976, pp.492-497.

[Wiederhold:92] Gio Wiederhold: "Mediators in the Architecture of Future Information Systems"; *IEEE Computer*, Vol.25 No.3, March 1992, pages 38-49;

[Wiederhold:95] Gio Wiederhold: ``Modeling and Software Maintenance"; in Michael P. Papazoglou (ed.): OOER'95: *Object-Oriented and Entity Relationship Modelling*; LNCS 1021, Springer Verlag, pages 1-20, Dec.1995.

---------- o ----------

Filename: ProductFlowModel.doc
Directory: F:\MyDocs\SW\ProductFlow
Template: G:\Documents and Settings\Gio Wiederhold\Application Data\Microsoft\Templates\Normal.dot
Title: The Product Flow Model
Subject:
Author: Gio Wiederhold
Keywords:
Comments:
Creation Date: 5/14/2003 11:36 AM
Change Number: 8
Last Saved On: 5/14/2003 6:17 PM
Last Saved By: Gio Wiederhold
Total Editing Time: 227 Minutes
Last Printed On: 6/1/2003 11:12 AM
As of Last Complete Printing
Number of Pages: 9
Number of Words: 3,580 (approx.)
Number of Characters: 20,408 (approx.)