The Product Flow Model

Gio Wiederhold
Stanford University
30 December 2003

"*It is not the strongest species that survive, nor the most intelligent, but the most responsive to change*" [Sir Charles Darwin, 1871]

Abstract

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

To sustain software in an enterprise requires several categories of maintenance. Today enterprises acquire most general purpose software from outside suppliers, rather than writing such software in-house, reducing total software acquisition costs. This paper considers how maintenance of acquired software should be provided. When the enterprise assigns maintenance of acquired software to its internal information technology (IT) operations for software then little overall long-term cost reduction is been achieved, since over its lifetime the maintenance of software consumes 70 to 90% of total software costs. An alternative is that the software supplier also provides the ongoing maintenance to the customer. The PFM defines in more detail such a business model that is beneficial to the supplier as well as to the customer.

For the product flow model to be effective 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, each requiring distinct feedback mechanisms. These mechanisms represent control loops inducing system stability. Software suppliers 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 we find individually crafted software, where an individual has a problem, formulates it, devises a program, runs and tests it, and walks away with the result [Dijkstra:72]. At the other extreme are enterprises that outsource all their computing needs to an outside services, perhaps to webservices via the Internet [Benioff:02]. Levels that may be delegated include system operation, development, and even management. The degree of delegation ranges from total control to limited, contractual control [Chou:03]. Most practical situations fall in between; unless the software embodies valuable intellectual property of an enterprise, its control can be

ceded by outsourcing, and decision becomes one of economics. Large enterprises develop software in their own Information Technology (IT) departments, but also depend greatly on purchased software. Managing the total software inventory so that the IT department is responsive to the diverse and changing requirements of an enterprise is a great 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. We will present this background in this ection before proceeding to the PFM itself.

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 an existing software module inventory to rapidly create semi-custom products for new customers. An example might be a company producing payroll programs for medium and large enterprises. Customer enterprises may 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 in generalization and documentation, 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 to customers that have matching requirements.

Reuse is enhanced by having a software architecture that allows effective reuse of the modules in the inventory. An 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 existing modules, inventoried modules that are modestly modified but retain similar interfaces, and wholly new modules.

The Product Line Model encourages companies that produce custom software to fund a reuse process 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 typically 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 information 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.

Responding to feedback generates maintenance work [Basili:90]. As stated by Barry Boehm [Boehm:81, p.533]:

> ".. The majority of software costs are incurred during the period after the developed software is accepted. These costs are primarily due to software maintenance, which here refers both to the activities to preserve the software's existing functionality and performance, and activities to increase its functionality and improve its performance throughout the life-cycle"

Successful software products have many versions, long lifetimes, and corresponding high maintenance cost ratios over their lifetime. Lifetimes before complete product (not version) replacement is needed are 10 to 15 years, and is likely to increase [SmithP:2000, Wiederhold:95]. Maintenance costs of such enterprise software amounts to 60 to 90% of total costs [Pigoski:97], [Jones:98]. Military software has been measured to be at the low end of the maintenance cost range, but that may be because is users don't complain much, and most feedback they generate is ignored. The effect is that military software is poorly maintained, and requires periodic wholesale replacement [Hendler:02].

The high level of effort needed for maintenance frustrates many IT shops [Swanson:76]. Unfortunately, many books covering software engineering, both for education and reference, omit the topic altogether [Jones:98]. Others propose naïve solutions, as tools to help purchasers to write better specifications and exhort software engineers to make their products more reliable. Those notions only address the problems that occur at the initial delivery, and ignore subsequent feedback, the essential component to sustain system stability [DoyleFT:92]. Maintenance addresses the issues that become known after initial release of a sustained product.

2. The Product Flow Process

An enterprise acquiring Product Flow software delegates software maintenance to the vendor of the software. The Product Flow Model focuses first of all 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 customer's 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 [Aron:83]. 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 the start of the process, 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 sustaining software supplier. Other, more detailed lists of maintenance tasks have been provided [Jones:98], but those tasks can be grouped into the three categories above, which distinguish the tasks by motivation, and later in this paper, by timing and feedback mechanisms.

There are two aspect that motivate a supplier to provide a high level of maintenance: one is to keep the customer -- it is easier to keep an old customer than to gain a new one -- and the other is income -- good maintenance has a high value to the customer and can and should be reimbursed.

2.3 Paying for maintenance

We find that PFM suppliers charge basic annual maintenance service fees on the order of 15% of the original license purchase costs. Some fraction of that will be used for minimal customer assistance, as help-line availability. Higher levels of support, for instance to provide on-site help, are generally available at higher prices. 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 will 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 a database systems vendor produced a better product, fully compatible with the published SQL standard. The supplier tried to use it 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:** We have found that software for a given set of functions, as measured in lines-of-code, grows approximately linearly over time. A rule-of-thumb is that a new version of a software product adds about as much code as the first delivered version contained. Larger increases cannot be managed. Small increases may not deliver enough value to justify reinstallation and retraining of staff at many customer sites. However, as software versions increases in size the interactions among their parts increase and the staff required to write such software becomes disproportionally larger. Since later, perfected

versions will be larger, a much larger staff is needed to create subsequent versions. [Brooks:95].

In practice, rewriting operationally useful software is becoming less and less feasible. The replacement of software that could not comply with Year 2000 problems may have caused 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. We now look at the information flow and feedback for the sustaining functions introduced in the previous section in more detail. The business model of the software supplier most support these flows if the PFM approach is to be effective.

3.1 Corrections

For corrective maintenance a supplier will make a reporting method to obtain feedback from the customer. A specialized team at the supplier will log error reports, filter stupidities, eliminate misunderstandings, combine errors that seem to be identical, devise workarounds if feasible, advise the customers, and forward any remaining significant problems to the engineering staff. The engineers, since the supplier operates in the PFM that values continuing improvement, will include the original developers. Fixing bugs now becomes an opportunity to learn, rather than a task to be dreaded. The focus in debugging is on the software itself, perhaps identifying and eliminating maintenance-prone software features [Arthur:83]. The next release will be improved in well-considered way. Once software is has been accepted by multiple customers over some time the effort needed for correcting errors is reduced, typically consuming less than 10% of the total maintenance burden [LientzS:80].

When reported errors are critical for some customers shipping them patches will be needed. That process is costly and disruptive. Although [Arthur:83] reports that 50% to 80% of software lifetime costs are due to maintenance, it is clear that he misses the boat by ignoring factors that are externally mandated and will not be mitigated by eliminating maintenance-prone software.

3.2 Adaptations

Adaptive maintenance is required to sustain software keep up with external changes that affect the supplied software. Aspects of our ever-changing world that affect software include:

- Changes in the computational system setting: new hardware, new storage devices, new operating systems. The customer is driven to gain benefits from new technology, and does not want to be held back by limits imposed by acquired software.
- Changes in communication: again, adaptation to upgraded communication protocols and methods is essential for survival. Failure to communicate well deprives the supplied product of input or renders its output useless.

- Updates of software that the supplied product must interact with at a customer site. 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
- 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.

Ignoring such changes can sometimes be fatal to a supplier, but delays are least painful.

Input for required adaptations comes from hardware vendors, from software vendors who make products that interface with the suppliers product, from government sources, and from standards bodies. A supplier operating in the product flow model will have representatives at many of these external vendors. A supplier 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 to their collaborators. In any case, a specialized PFM software supplier is much less likely to be caught unaware than an enterprise IT department, and may have enough clout to influence specifications and deadlines.

3.3 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 by the supplier. It certainly should not require retraining at the customer. Improved functionalities are natural upgrades, they follow the same paradigm, an often make 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 or 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 hinder ongoing customer operations.

Making the upgrades 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 [Chou:03]. Serving an existing customer will also bring much realism into the request flow.

3.4 Development approach.

All three types of inputs come together in the engineering staff, and drive the development of successive versions.

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 work incrementally, deal with rapid turnaround, and maintain a 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, avoiding 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 an instance where academic education fails. Students in our schools are taught immediately to write programs, and rarely exposed 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 can reinforces the switch to `read before you write'. Focusing on smaller code segments reduces the risk of serious failure.

4. Economics

For the supplier, retaining customers, and in effect converting the sales-oriented business model to a sustaining product flow service model has substantial benefits, and greatly changes the long-term cash flow. In Figure 1 we show a simple model, where sales of a software product increase over 5 year, remains steady for 10 years, and then fall off. Derived maintenance license revenue, estimated at 15% of license fees and an annual loss of existing customers of 10% brings in 50% more income than the gross sales, albeit with a substantial lag - about 50%, requiring some discounting to current value [Wiederhold:83, Chap.6.4.2]..

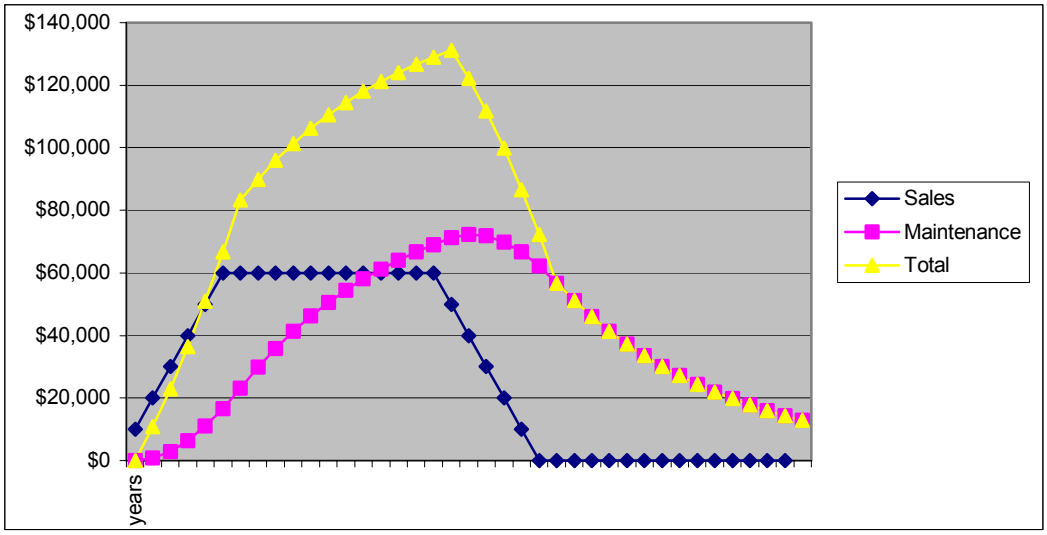

Figure 1. Gross income from sales and maintenance

Figure 1 is based on gross revenues. In practice the net income from sales is less, since much more effort is involved in making a new sale then it is to renew a maintenance

contract for another year. We find that net income from sales licenses is about 50% of gross while net income from maintenance licenses may be 75%. On the other hand, since maintenance revenues are delayed, their effect is less in terms of current value.

Not obvious from this figure are the ongoing development costs that are now covered by maintenance income. Since sustaining maintenance leads to substantial product perfection, a large fraction of development costs are now embedded in maintenance. Customers in later years receive a substantially better product for the same price. New product development, or adding substantial new functions, still require investment, but leads then to new products that can enter a similar sales and maintenance cycle

An extreme case of focusing on potential service income is offered by the supplier and maintainers of open-source software. Unfortunately, many purchasers of open-source software are inadequately aware of the maintenance needs that they will have. Since the dominant open-source software licenses stipulate that maintenance improvements will be made freely available by all for all there is no direct economic motivation to deliver consistent adaptation or perfection. The assumption that we will have enthusiastic programmers generating open source software may apply more to developing novelty than performing maintenance. Developing a workable business model that allows benefits to accrue to both the supplier and the customers is a challenge [Johnson:01]. I don't have enough data on open-source software providers to project the future of open source maintenance suppliers, and the situation is now fluid [McMillan:03].

Other service income

In addition to revenue from maintenance licenses software companies also benefit from consulting and education services. These sources may well be in conflict with the objectives of the product flow model. Generating more consulting service income because the product is inadequate is an obvious danger to operating successfully in the product flow model, it indicates that the product is unsuitable in some way [Moore:95].

In the product flow strategy adapting means dealing with new external requirements of general applicability, not customizing the product for some specific customer. If a specific adaptation is needed it should be seen as a potential generalization. If it is not, it indicates a mismatch of the product with the customer's objectives, and such while adaptations generate immediate consulting income, they do not provide the long-term stream of income that sustaining a product flow should provide. If programming resources are scarce it will be wise to focus on one primary business model.

Perfecting includes making the product easier to use for all customers and thus reducing training requirements. Training customers to set up or use a complex product may well generate education income. Such income has low margins since customers will not want to pay much for services that only delay their ability to profit from the product. For the supplier providing good educators is costly. Perfecting the product so that less education is needed is a better long-term strategy.

5. Management responsibilities.

Moving from a product-line approach to a product flow approach incurs management costs. Technical as well as marketing and sales divisions must be refocused. There have been instances where companies desired the income from continuing services, but did not change their business model adequately. This problem has also been observed at many enterprise sites [Brynjolfsson:98].

5.1 Technical staff adjustments

To motivate the technical staff it is crucial that the maintenance tasks be highly valued. Today computer science education and many organizations value novelty above all other aspects of software development. Sales and marketing personnel often push `new' features over reliability. The term `maintenance' itself in such a setting is then perceived negatively by the software engineering staff [Jones:97]. Assigning new hires to maintenance, as is now often done, diminishes the value of sustaining improvements. Experienced engineers are actually more likely to have the needed know-how for the maintenance tasks outlined here. New, bright hires may be better suited to developing new products and functions.

Costs will be incurred in garnering the input for timely adaptation of the product. Participation in standards efforts can provide early indications of required adaptations, keeping products fully competitive. Participation requires long-term commitments by some valuable technical people, but is relatively easy to accomplish.

5.2 Marketing and sales staff adjustments

Obtaining effective input from customers in order to support perfecting requires a broader and more difficult conversion in the marketing and sales arena. New communication channels have to be created to transfer information from sales and marketing to the engineering staff. Sales and marketing people must learn to listen to customers, rather than pushing existing products. Features of competing products should be appreciated, not put down. Eliciting long-term customer needs is not easy, but valuable to set directions for technical investments. The needs of the customers must be aggregated and presented so that product objectives are clear to the engineering staff.

The management of the software supplier itself must support this model throughout all layers and divisions of a company. The change of emphasis may be hard to adopt by managers that were raised, and became successful with a focus on product development. The income benefits brought by a product flow business model to the supplier from supporting the maintenance services do not come without structural changes.

5.3 Moving from Hardware to Software Income

Many companies that traditionally have focused on computer hardware find the software market attractive. The commodization of hardware has lowered its margins. Software products promise to provide the income that is associated with a higher intellectual property content and a longer life. For such a corporate transition the conversions is even greater. Since hardware is for practical purposes unmodifiable, it is characterized by a short lifetime; say 3 years for a personal computer, and less than that for handheld

gadgets [KimS:03]. In that setting maintenance is either ignored or given very low status. However, any software should live for a long time to garner maturity, market penetration, and the resulting benefits. A major corporate attitude shift is needed to accomplish the transition. However, management, often unfamiliar with software production, will have a hard time directing such a transition. Similar observations have been made on the utilization of IT to gain productivity [BrynjolfssonH:98].

6. Conclusion

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

If the software makes the suppliers' customers more successful then customer growth will cause unit sales to increase. Many software licenses have provisions for charges proportional to the number of active users. Existing customers are also prime targets for new, related products.

New customers of the old product automatically receive the current, improved product, and are integrated into the ongoing PFM process. The effort to obtain input for product requirements and development costs for prior upgrades were, at least in part, supported through the sustaining maintenance process.

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.

Moving to the PFM model requires a management commitment and restructuring of technical and management functions. One cannot garner the benefits without such an effort. Little financial investment is needed if personnel can be motivated

Acknowledgements:

The term `product flow' was coined by me, but owes its existence to other software engineering researchers, especially Barry Boehm. Many of my students provided input to these concepts, especially David Liu. Software companies that receive substantial from support can look at their operations and decide to what extent their business matches the model presented here. Many companies are desirous of gaining more income from support, but do not have all the components in place. My work is supported through the MITRE, a Federally Funded Research and Development Corporation, as part

investigations into the value of corporate intangible property. I have to thank many unciteable sources for motivating this work and providing data for its validation. Information on income from support revenues is included in many corporate reports, but requires cross-referencing and analyses since it is rarely reported in a consistent fashion. An early version of this paper was presented as a keynote at the 2003 SEKE conference and led to useful feedback [Wiederhold:03].

References:

[Aron:83] J.D. Aron: *The System Development Process, The Programming Team (Part II)*; Addison-Wesley, 1983.

[Arthur:83] Lowell Jay Arthur: Programming Productivity; Chapter 8 on Maintenance, Wiley, 1983.

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

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

[Benioff:02] Marc Benioff: "Salesforce.com: What Economic Downturn?; *Destination CRM,* January 2002, www.destinationcrm.com/articles/default.asp?ArticleID=391.

[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.

[Brynjolfsson:98] Erik Brynjolfsson and Lorin Hitt: "Beyond the Productivity Paradox"; *Communications of the ACM*, Vol. 41, No. 8, August 1998, pp. 49-55.

[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.

[Chou:03] Tim Chou: "The Hidden Cost of Software"; *ASP news*, www.aspnews.com/strategies, May 29, 2003.

[Dijkstra:72] O. J. Dahl, E. W. Dijkstra, C. A. R. Hoare, *Structured Programming*, Academic Press 1972.

[DoyleFT:92] John Doyle, Bruce Francis, Allen Tannenbaum: *Feedback Control Theory*; MacMillan 1992.

[Hendler:02] James Hendler et al.: Report on Database Migration for Command an Control; United States Air Force Scientific Advisory Board, SAB-TR-01-03, Nov. 2002. .

[Johnson:01] Justin P. Johnson: The Economics of Open Source Software; MIT, http://opensource.mit.edu/papers/johnsonopensource.pdf.

[Jones:98] T. Capers Jones: *Estimating Software Costs*; McGraw-Hill, 1998.

[Jones:97] T. Capers Jones: *Software Quality -- Analysis and Guidelines for Success*; International Thomson Computer Press, Boston,1997.

[KimS:03] Sang-Hoon Kim and V. "Seenu" Srinivasan: A Multiattribute Model of the Timing of Buyers' Upgrading to Improved Versions of High Technology Products; Research paper 1720R, Stanford Graduate School of Business, 2003.

[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] john J. Marcianak *Encyclopedia of Software Engineering*, Wiley, 1994.

[McMillan:03] Robert McMillan**:** Red Hat users balk at Enterprise Linux licensing; *IDG News Service,* 11/10/03.

[Moore:95] Geoffrey A. Moore: *Inside the Tornado : Marketing Strategies from Silicon Valley's Cutting Edge;* Harper Business, 1995.

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

[SmithP:00] Gordon Smith and Russell Parr: *Valuation of Intellectual Property and Intangible Assets*, 3rd edition; Wiley 2000 (p 304)

[Wiederhold:83] Gio Wiederhold: *Database Design*; Mc Graw-Hill, 1983, also available in the ACM anthology, 2003.

[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.

[Wiederhold:03] Gio Wiederhold: "The Product Flow Model"; Proc. 15th Conf. on Software Engineering and Knowledge Engineering (SEKE), Keynote 2, July 2003, Knowledge Systems Institute, Skokie, IL., pp. 183-186.

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