Determining Investment Lag

Gio Wiederhold

Stanford University CSD

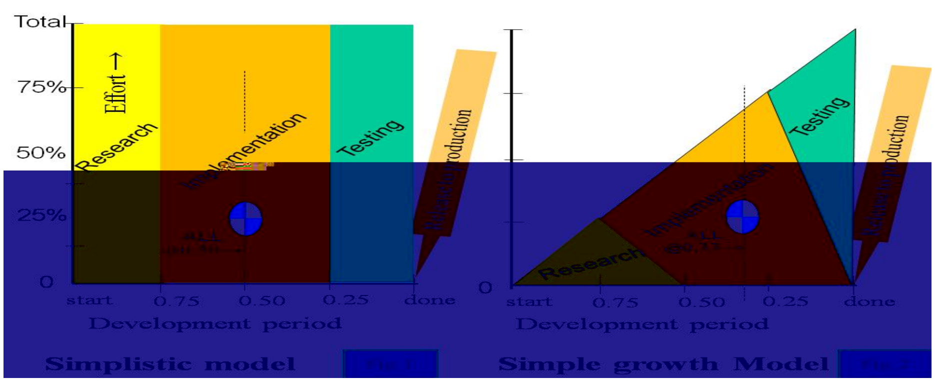
San Francisco, March 2008

1. Introduction

This paper describes methods to compute a parameter, investment lag, useful for relating financial metrics to product development. Investment lag, as considered here, is the delay between the time that R&D investments are made and the time that revenues or equivalent benefits are realized. The context of the analyses shown in this paper is software development, but the methods are not necessarily limited to that application, since lags occur in many contexts. In the general business context, the lag we address is one specific form of operational lag [McConnelB:05]. We focus on cases where investments are made over several years, so that computing a single parameter, the effective lag, requires an understanding of investment patterns. We also consider the common case, where software continually evolves through many versions, and investments continue. To arrive at guidance we develop models that parameterize investment and effort components during software development. Given the assumptions stated with the various model types, the results are obtained in a straightforward manner. The major contribution of this paper is in bringing together the issues that affect lag during product development and providing consistent models for discussion and evaluation.

1.1 Definition

Investment lag denotes the interval between an investment and the time when that investment first provides benefits (*h* in [BarIlanS:96]). Product development requires ongoing investments over some time, sometimes called the economic gestation period, often spanning several years. In order to aggregate the incremental investments over the development period into one parameter, the effective lag expresses the average time from investments until the time that development is complete. We depict the effective lag time relative to project completion by a symbol (◐) which denotes the weighted average, or centroid of the investment pattern.



In the remainder of this paper, we refer to the effective investment lag simply as lag, although the term lag has other meanings in other settings, as cited in Section 1.4.

1.2 Use of lag

The benefit of the concept of lag is that a simple parameter can be used to characterize a complex issue, namely the financial effort over time expended to develop or upgrade a product.

Working capital estimation. In a business plan, having a lag estimate helps in judging requirements for initial and ongoing working capital. The tradeoff between development pace, development risks, and benefits of early product release to sales require a clear model.

Capitalization of R&D investments. Once it is considered probable that a software project will be completed and put to its intended use, the ongoing development expenses for software having an expected life of more than two years should be capitalized [FASB:85]. Having a model of lag helps in planning of software expense capitalization and amortization decisions. Today FASB guidance is often ignored, although capitalization of research could be financially advantageous and help shareholders understand corporate actions [MulfordR:06].

Version release patterns. Software is characterized by its adaptability to new requirements. Any successful software product will have versions that supersede earlier releases, at intervals from one to several years. Over time the effort involved in evolving successful original software for subsequent version development exceeds greatly the original effort [Pflegger:01].

Software Re-creation. Sometimes software must be recreated. One reason may be having a legal problem, as when software has been inadvertently appropriated from an original creator that denies the user the right to that software. An important case was Fujitsu's use of IBM's OS/360 after IBM removed its updates from the public domain [Jussawalla:92]. Estimating the lag incurred in recreating software can be crucial.

Software Valuation. Lag is important when estimating the intellectual property (IP) value of a software product since the lag has to be accounted for in allocating income from the software development investments [Wiederhold:06]. When a multi-version software product has to be valued both the initial development lag and the version development lags have to be considered.

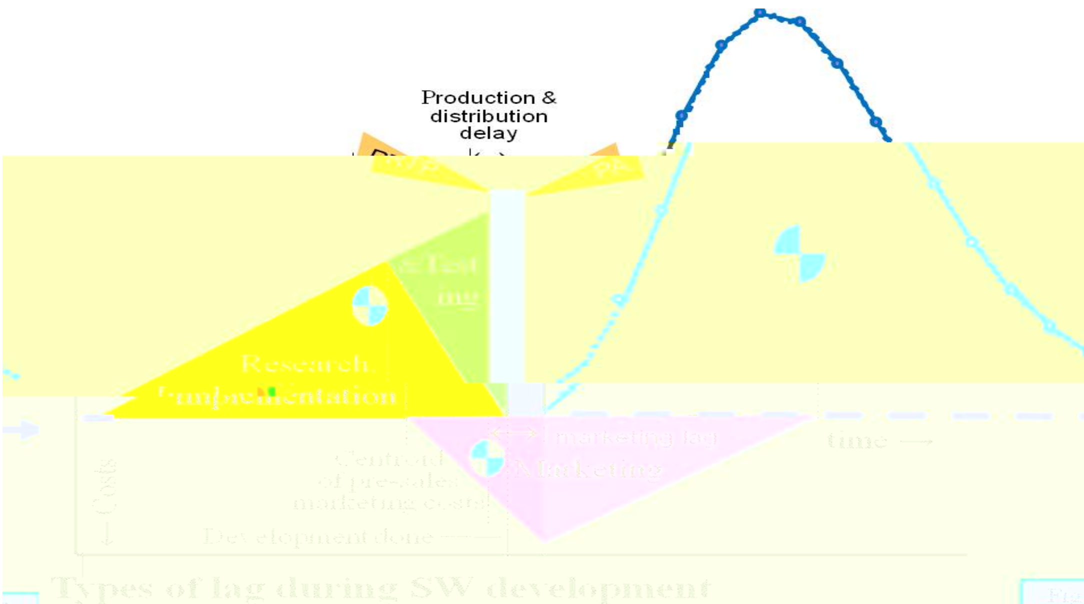
Risk estimation. Development risk cannot be quantified when the investment pattern is not known. When undertaking any project the risk of success must be tracked. Divergences from planned effort rates and changes in lag for development components are an indication of problems. For instance, offshoring of quality control can reduce personnel costs and allow testing to be carried out interspersed on a daily cycle with ongoing product improvements. Risks associated with losing control of IP when offshoring requires balancing the value of the software with cost and time savings that can be achieved.

1.3 Simple Models of Lag

For the simplistic case shown in the introductory Figure 1 the rate of spending is constant. If the development takes three years, the effective investment lag is 18 months. Expressed as a fraction over the development time, that lag is 0.50. Indeed, we often see that in technology development projects the effective investment lag is being estimated as half of the interval from development start to project completion. Such an over-simplification easily leads to surprises in budgeting.

A simple but more realistic model, based on a linear growth of the development effort, was shown in Figure 2. In that case the effective lag, computed as the centroid of the effort levels and the time spent, becomes 0.33 of the total development time. Since the date of product completion is the crucial point, we measure the effective investment lag as a fraction of the development period relative to that final date. Effort levels are measured in terms of cost.

1.4 Varieties of Lag



The definition given in Section 1.1 applies to the total development lag, but the term lag has also been used for other efforts in product development and sales. We will briefly cite them here, before returning to the main focus of this paper, investment lag during development.

As shown in Figure 3, subsequent to the development lag, the point when the product is ready for release to production (RTP), there will be a delay to account for actual production and distribution before product availability (PA). Only then sales can commence, characterized by a sales lag, In parallel, there will be marketing investments, with a marketing lag.

The effective sales lag is computed from product availability to the centroid of the income during the period where revenue is generated from the product. Another term for that metric is average sales life. It is also a critical metric for financial analysis, and affected by many considerations [Klepper:96]. For software developed for internal use, metrics of internal benefits would replace sales and their revenues in these discussions. Sales lag is not addressed further in this paper.

Marketing-specific investments might start when the product is adequately defined, say when testing commences, and continues from that time on.

The term lag has also specific meanings in other settings. For instance, in project management, lag is the interval between tasks [PMI:04]. The term lag has been used when comparing large-scale investments, indicating that some company or country has not invested as much as its rivals over some extended period of time.

1.5 Manufacturing and Distribution Delay.

Tangible products incur delays between RTP and having a sufficient inventory for sales to commence. Modern production methods have greatly reduced the delay for production and distribution. For software products distribution CDs can be replicated and packaged in days. Distribution of intangible products or services over the Internet incurs even less delay. Delivery of products as PCs with pre-installed software can take a bit longer, but is still negligible compared to development times.

For software the date of Product Availability (PA) can be set equal to the date of release to production (RTP), simplifying the overall model by omitting the production and distribution delay interval shown in Figure 3. We can then refer to the merged points as release to sales (RTS),

1.6 Marketing Lag.

For software and other products that do not incur a significant production or distribution lag, the marketing lag becomes the period prior to RTS. A mature [Damodaran:05] and well-known organization can effectively market new products and new versions well before the development period is over. Overall, marketing efforts prior to RTS are relatively small in the software industry, but can differ greatly. Major product announcements by a well-known player about introductions, updates, and replacements can generate much hype even without large investments.

While detailed market planning may start soon after research is completed, marketing efforts become costly only during the latter parts of development and testing. Excessive early visibility will impact ongoing sales of any prior products or versions. But when beta releases are made available to experts and trusted customers during the testing phase they will create a marketing buzz and marketing must get involved. Some time prior to RTS additional marketing expenses will be incurred. Marketing lag should be small for products with a well understood functionality. As products evolve and new versions are released additional marketing costs are incurred, to encourage customers to upgrade [AmblerC:96].

After RTS marketing costs continue, but are no longer considered investments towards future revenues. Such marketing costs are best considered costs associated with sales, and do not contribute to the marketing lag estimate. Marketing costs for evolving software products in terms of total revenues tend to average out to about 6%, while development costs average annually 11% [Desmond:07]. These costs vary widely depending on product types and customer base. We do not deal with marketing lag specifically, but because of its linkage to development lag the models in this paper contribute some information to the issues.

2. Development lag

We now focus on development lag, considering its components and its metrics.

2.1 Effort components during development

The type of effort needed to develop a product differs over time. As shown in Figures 1 and 2, product development requires

1. *research*,

2. *implementation*, and
3. *testing* efforts,

each contributing value to the product.

In practice there is not a stable level of effort nor a rigid boundary for these component efforts. The combination of research and implementation is often referred to as R&D, but in this paper we reserve the term *development* for the total effort, including testing. Researchers may join the implementation teams, or remain separate, just inspecting the implementation to validate their concepts. We consider activities that lead to a specific product, as program design, task planning, coding, and unit testing as part of implementation, but keep product testing distinct. Typically, testing for quality assurance is carried out by distinct teams that deliver their findings to the engineers working on the implementation. Adequate testing requires 40 to 50% of the total development effort [Graham:94].

The three components differ in terms of personnel requirements, risk, and criticality. Cost of required personnel and support will differ for these effort components. Conversion of costs to headcounts is not done within this paper, but recognition of these components enables the needed mapping. Once all research is completed less risk remains. The initial design, which should be completed within the research component, influences greatly all subsequent implementations [Lammers:86, p.76+]. Early implementation efforts also reduce the risk, eventually allowing capitalization of the expenses. In this paper we compute estimates for the centroids of each of the three components, but do not try to assign weights to their costs and contributions.

2.2 Metrics

The most common metric for the effective lag is time, the number of months corresponding to the weighted average of incurred investments. But software development durations can range from many years to a fraction of a year. In order to decouple lag from the size of the project we use in this paper the fraction of the product development time, with 1.00 to indicate the start of the project, and 0.00 its completion, as shown in Figures 1 and 2. Development terminates when the product is ready for release to production (RTS), and no more changes can be made. Throughout the development period, costs are incurred. In Section 4 we consider the development of successor versions, and we again use fractions of the interval between each version release.

Development costs are best determined from internal data on expenditures for development and for acquisitions of elements and tools that contribute to the product. R&D expenses as booked typically include testing. Sometimes staff work-hours are used as a surrogate, but then one should recognize that people involved in research and early development may have high pay rates, whereas people involved in testing may be paid less per hour. Marketing lag inputs include costs of pre-sales promotions and advertising. Sales lag can be obtained from detailed revenue data.

However, this note focuses on lag estimation, where actual data are not available. The estimation models themselves can be validated with actual historical data.

3. Types of Development Lag

Lag differs of course based on the size of a project, but also on the setting where the development takes place. For a startup the total elapsed time to have a saleable product typically ranges from 2 to 5 years. Even a product developed in a mature organization, with experiences staff and management to draw upon, will be at least a year.

3.1 New Products, Versions, and Replacement Products

Lag patterns for successor versions of a product differ from the lag encountered when developing new products. New products have substantial research risks, as well as unknown competitive threats requiring adaptation and flexibility. Minimizing lag is crucial, but made easier by not having to cater to an existing community.

For successor versions the lag depends on the extent of updates (changes and additions) required and interaction of updates. Research is less, but testing to assure compatibility increases. As product versions become more complex, lag is likely to increase, unless changes to successor versions are minimized. The effort of testing the interaction of new code with all remaining code in a version depends also on the size of the prior version. If changes become too great, it becomes impossible to have a reliable successor version [Bernstein:03].

When a new version is released, prior versions have to be maintained for substantial periods, until customers shift to the new version. Maintaining overlapping versions incurs substantial costs with few benefits for the manufacturer.

Obsolescence of the product's design may dictate the creation of a replacement product, where none of the old code is reused. Now the lag is much greater. For a replacement all usage patterns, documented and undocumented, of the prior product have to be checked, lest the replacement fails to satisfy customers' compatibility expectations. Maintaining overlapping products incurs substantial costs and introduces market confusion with no benefits for the manufacturer.

If writing replacement software is mandated by financial or legal concerns, the magnitude of the lag is even greater. The organization wishing to replicate the functionality of the software may not have access to the original code of the original product being replicated. Internal documentation would also be protected. That organization also cannot exploit the knowledge that resides in the minds and memories of the original authors. Only the external, public manifestation of the software is available. Even the effort to create adequate documentation can be substantial [WiederholdE:71].

3.2 Factors for determining lag

In Figure 4 we show a simple model. The validity of the results of a model will affected by several conditions:

What is the maturity of the organization which is developing the product?

1. What is the state of the product: is it novel or an improved version of a prior product?
2. Is the number of developers working on the product growing, or relatively stable?

Furthermore, if the development is to be analyzed in more detail:

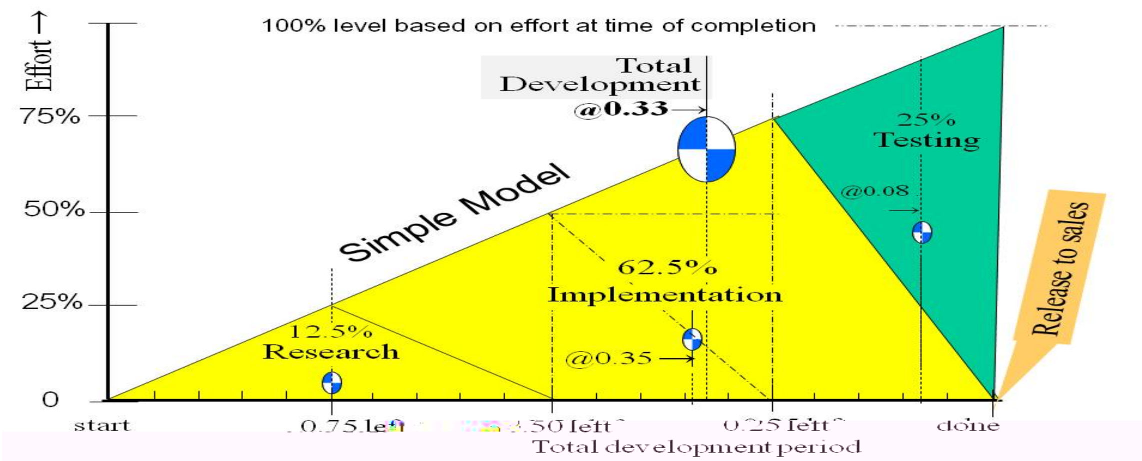
3. What are the efforts to be spent for research, implementation, and testing, in terms of effort and period needed for those components?

The level of effort typically increases as the product is closer to being released for sales, but the curve of effort is often complex.

Since acquisitions can represent major increments outside of the parameters of the model they will not be considered in the model analyses. If an acquisition brings in staff, and its history can be determined, that data can be aggregated into the model.

4. Initial Development lag

We sketch the range of efforts involved in bringing a new product to market in Figures 4 to 9. The total amount of effort spent grows steadily during the development period. The ratios of effort spent on research, development, and testing will differ among products and versions. In a mature company the effort can ramp more rapidly, while a company starting up will tend to ramp up more slowly.



Quantified model efforts during SW development

Fig.4

4.1 The Simple Model

Given a simple model, shown in Figure 4, for the components and timings of the development efforts, we can estimate the efforts and when they take place relative to release to Sales (RTS). The magnitude of the component effort distributions match software engineering experience. Following Figure 4 the total development effort is 50% of the maximum effort × the development time. The centroids for the effort components become

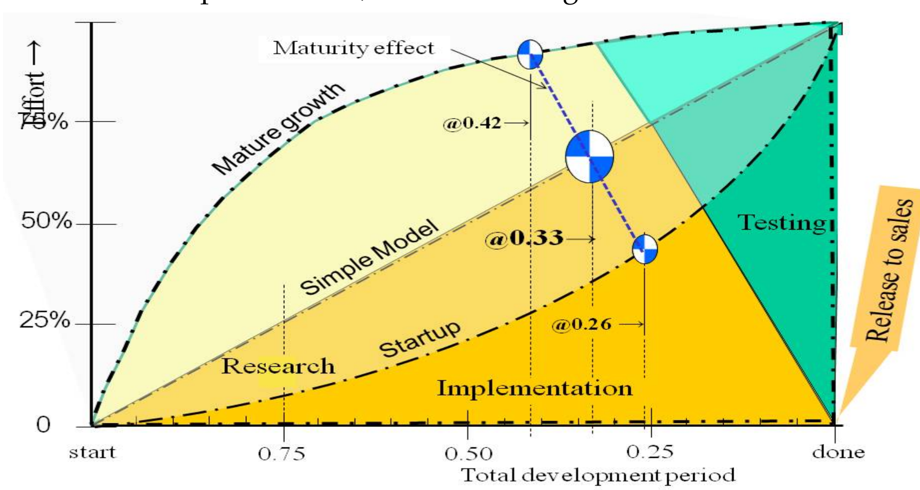
1. The total effort towards a new product has a centroid at 0.33 before RTS
2. Research: 12.5% of total effort, centered at 0.75 before RTS
3. Implementation: 62.5% of total effort, centered at 0.35 before RTS
4. Testing: 25.0% of total effort, centered at 0.083% before RTS

These results are obtained by taking the triangles that represent the components, basing their weight on their relative areas, and, for the centroids using the distances from each of the triangle's own centroids -- one third from their vertical edge -- to the RTS edge. For the Testing triangle of Figure 4 the centroid is at 0.25/3 + 0 = 0.083 before RTS. The Research effort is represented by two triangles of equal weight, with centroids at 0.083 + 0.75 and –0.083+ 0.75, giving 0.75 before RTS. The Implementation effort area can be obtained as the sum of 5 triangles (or 3 triangles and a rectangle), but is actually simpler to compute by taking the total effort weight minus the testing and research weights: 100%-12.5%-25%=62.5%. Its centroid is then computed by subtracting the weighted research and test centroids from the total centroid: (0.33-12.5%×0.75-25%×0.083)/62.5%=0.35 before RTS.

Often Research and Implementation (R&I) efforts are combined. In the simple model they comprise 75% of total effort, centered at (0.33-25%×0.083)/75%=0.42 before RTS.

4.2 Maturity Effect

The simple model ignores practical growth considerations. Personnel growth on a project is rarely linear. If no actual data are available we model a range of personnel growth as shown in Figure 5. For a project and organization that is novel, we expect slow initial growth, as indicated by the lower curve labeled Startup. Similarly, a mature company can rapidly grow the staffing of a project, as indicated by the upper curve "Mature growth" [Damodaran:05]. An effect is that the centroid of development shifts, as shown in Figure 5 as well.



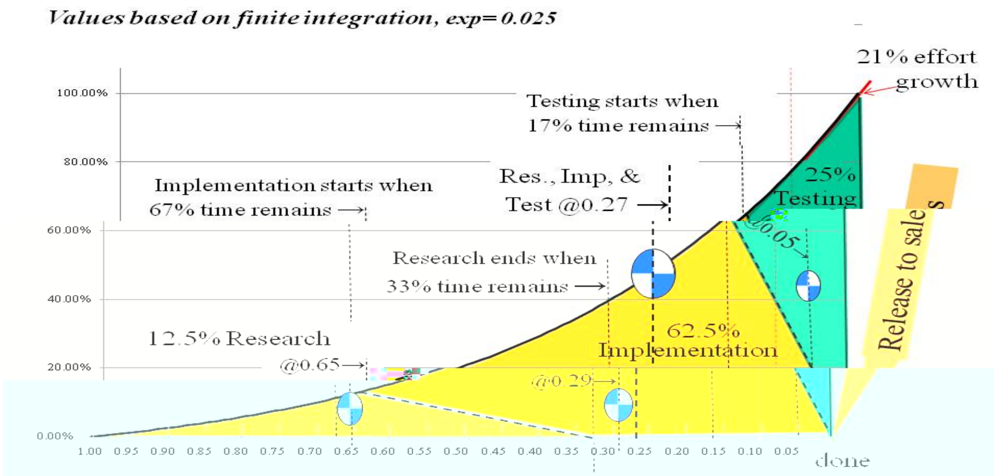
Maturity effect

Fig.5

The centroids shown in the sketch of Figure 5 are based on the detailed computational models presented in Sections 4.3 and 4.4 below. Research and testing centroids shift as well.

4.3 Initial Lag for a Startup

As discussed, a company just starting up typically does not have the resources to satisfy the simple model of Figure 4.



Start-up development with 12.5% research and 25% testing effort

Fig 6

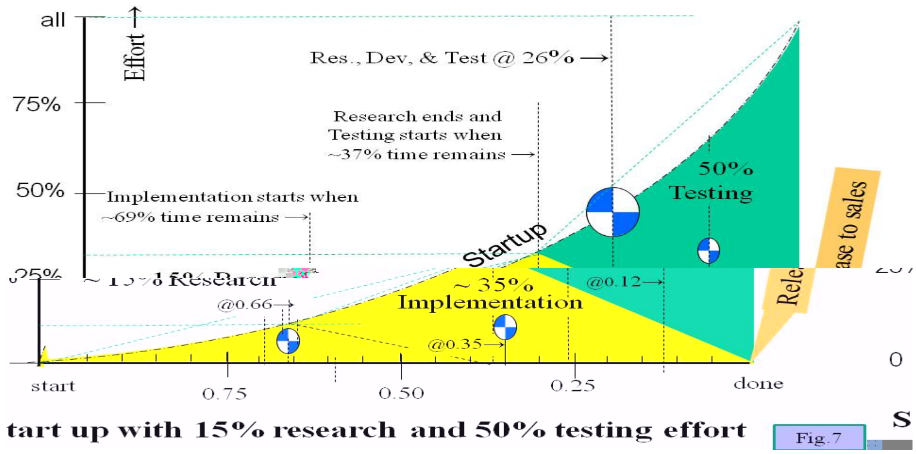
A startup company may have to work initially with very few resources, and only after it demonstrates feasibility can attract venture capital to move towards product completion. The total development time will be longer, but the centroids will shift towards the delivery date, as shown in Figure 5. For a more precise analysis we show the results based on exponential growth. The growth over the period was determined by limiting the effort growth during the final 10% of the development period to about 20%. Higher rates of adding and training personnel cannot be sustained in practice. Such a growth rate is achieved with an effort growth curve $Effort = fraction^{(1+x)}$ with x= 0.025, as shown in Figure 6 [Wiederhold:08S].

The results shown in Figure 6 were obtained by finite integration, providing a more accurate result than can be obtained by decomposition into triangles, although in practice simple geometric computations suffice.

For the startup case of Figure 6 the centroid for the entire development effort shifts to 0.26 of the development time before release to sales. The relative efforts have been kept the same as for the simple case, increasing the research interval from 50% to 67% of the development period, and delaying implementation correspondingly. The centroids for the three effort components become: research 0.65, implementation 0.29, and testing 0.05 of the development time before release to sales.

The effort devoted to testing allocated in Figure 6 is modest, as is typical for a startup.

Figure 7 sketches the case for a startup where 50% of the effort is devoted to testing. The overall development centroid remains at 26%. The estimates for the centroid positions of the three effort components, approximating the concave segments by weighted triangles, become: research at 0.66, implementation at 0.35, and testing at 0.12 of the development time before release to sales. Both styles of computations are available in the accompanying spreadsheet, [Wiederhold:08S]. While a mathematical approach such as integration appears to be more precise, the use of graphs conveys more understanding.

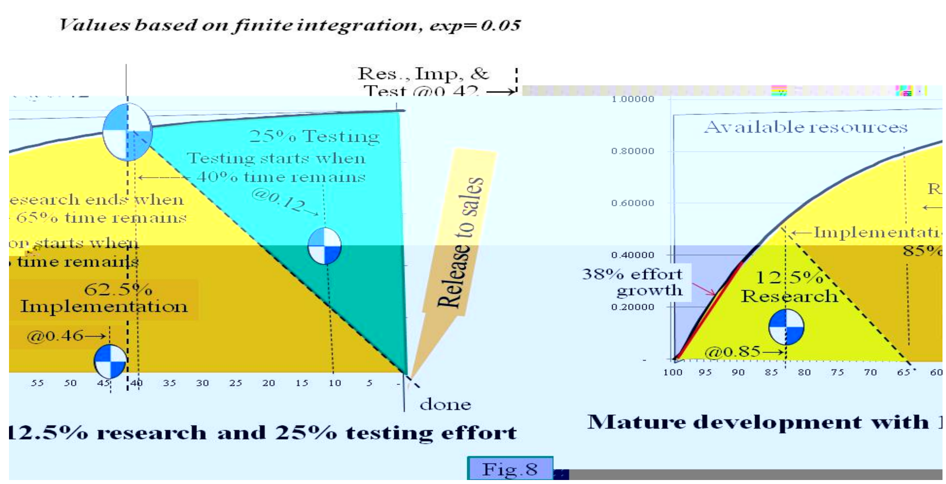


4.4 Initial Lag for a Mature Company

A mature company will be able to ramp a development effort for a new product more rapidly, in effect moving the centroid to the left. Such a personnel allocation reduces the total development time, but advances the relative lag in the interval. The lag fraction we use throughout will increase.

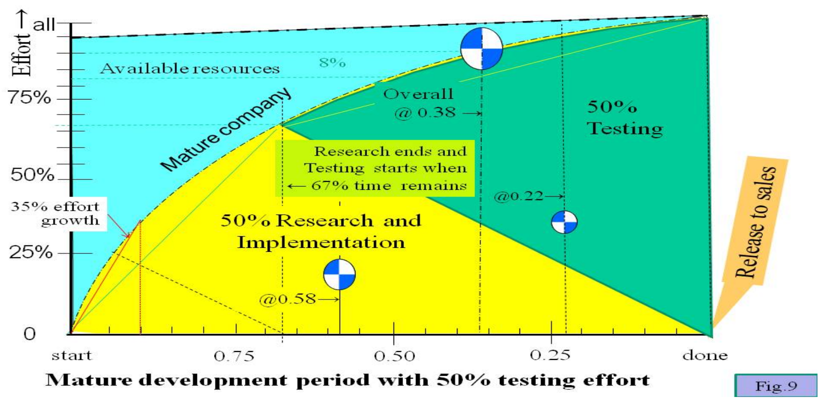
While the limiting criterion used for the startup model was personnel growth near the end of the development period, for a mature company the assignment of internal personnel is not subject to the practical growth limitation of external hiring. But the same factors should be considered.

To create a complementary model to the startup case for a mature company we reverse the curve used for the startup model. The expected effort growth in the initial 10% of the development period is now about 38%, including the 10% growth expected from the simple, linear model. The exponent now becomes 0.05. We use the same effort distribution as in the Simple model (Figure 4) and the initial Start-up model (Figure 6): 12.5% research, 62.5% implementation, and 25% testing. The result is shown in Figure 8. The effect is that the overall development centroid is at 0.42 of the development interval. The centroid for Research is at 0.85, the centroid for Implementation is at 0.46, and the centroid for Testing is at 0.12 of the development time.



Computing the centroids for research, implementation, and testing efforts separately is useful if distinct teams carry out these functions. The salaries paid to these classes of personnel can differ as well. Often the testing effort is outsourced to specialist groups. When it is hard to distinguish Research from Implementation efforts, they are best combined. The centroid for the 75% Research and Implementation efforts combined, not shown in Figure 8, is at 0.52 of the remaining development time.

In a mature company the need for extensive testing should be recognized, even for a new product. In Figure 9 we sketch the effort distribution for a case where testing occupies 50% of the total development effort. The initial growth was assumed to be 35%. The centroids are estimated based on an approximation using triangles. The effect is that the overall development centroid is at 0.38 of the development interval. The centroid for Research and Implementation combined is now at 0.58 and the centroid for Testing is at 0.22 of the development time.

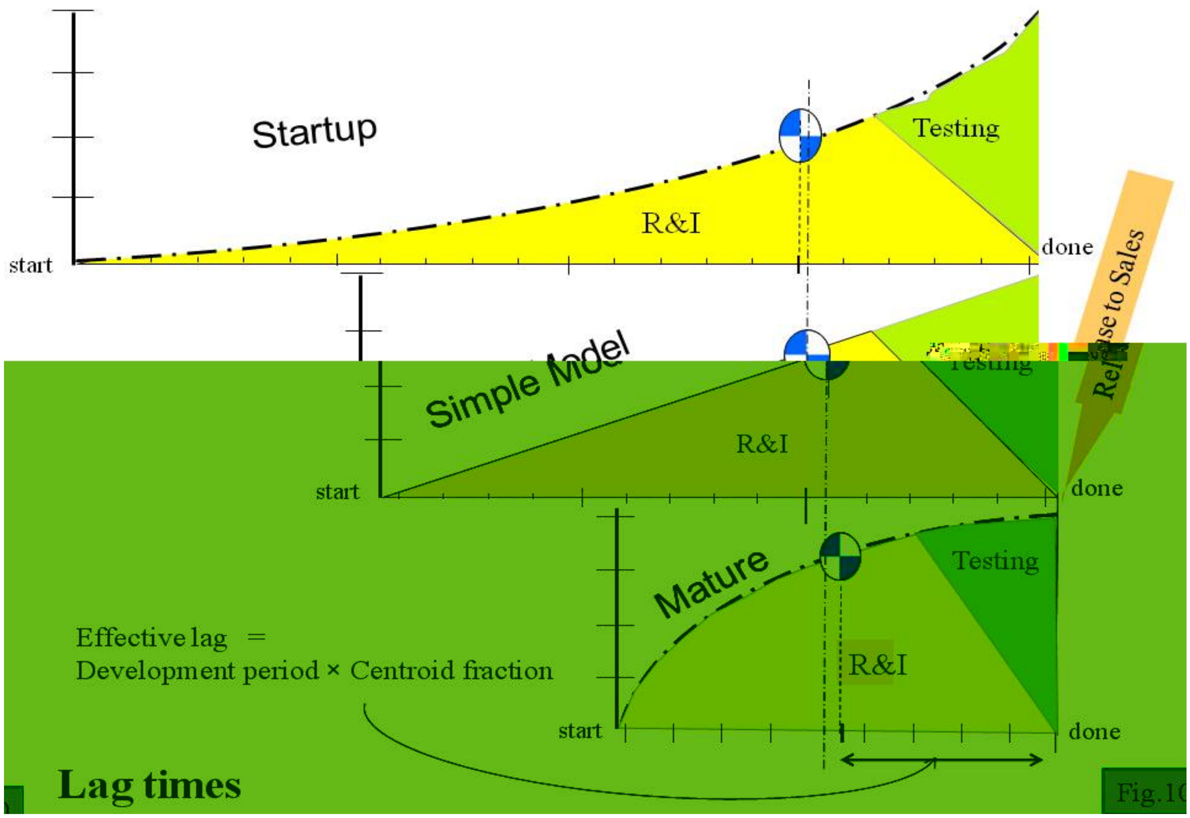


Developer maturity	Testing fraction	Total effort centroid	Research centroid	Implementa-tion centroid	Testing Centroid
Startup	**25%**	0.27	0.65	0.29	0.05
Startup	**50%**	0.26	0.66	0.35	0.12
Simple	**25%**	0.33	0.75	0.45	0.08
Simple	**50%**	0.33	0.88	0.43	0.17
Mature	**25%**	0.42	0.85	0.36	0.12
Mature	**50%**	0.38	0.58		0.22

Table 1. Typical initial development lag parameters and results

4.5 Lag centroid versus effective lag time

If the effort expended in the three maturity models is approximately equal, the actual, effective lag time may not differ much. In the examples of Figure 6 and Figure 8 the development efforts over the period were 63% and 158%=1/63% relative to the simple model of Figure 4. In Figure 10 we sketch the three models, scaled so that their actual areas become equal. We see that while the actual development time differs for the same development effort, the centroids for all three models are in similar positions. This observation can simplify the centroid estimation if only the amount of effort is known, but not the actual period over which the development effort took place.



While a mature company can organize its resources and shorten development time, in practice the productivity of staff differs as well. Assembling the workforce for a project is the most crucial aspect in software development [Glass:03]. Larger organization will have more bureaucracy, reducing individual effectiveness. Individuals in the initial stages of a startup will be very productive, but are rarely well prepared to deal with rapid growth on personnel. Training of new staff has a serious impact on existing personnel [Glass:03]. A well-known maxim is that adding personnel to deal with a late project makes the project later [Brooks:95]. The subsequent discussions in this section will refine the maturity models for ongoing development, creating new versions.

5. Version development lag

The ability to learn from ongoing feedback from customers is essential to any long-lived product. The flexibility of software makes insertion of IP from such feedback especially effective. During the life of a software product new versions will be released, with each version having substantial improvements in reliability, capacity, scope, and complementary functionality. This work encompasses the three recognized aspects of maintenance [IEEE:02]:

1. Corrective maintenance, i.e., bug-fixing
2. Adaptive maintenance, keeping products up-to-date with standards, platform updates, and communication improvements.
3. Perfective maintenance, as improving operation and usability, adding capabilities, assuring scalability, assuring smooth and consistent operation of the software, and dealing with security threats, all to match increasing customer expectations.

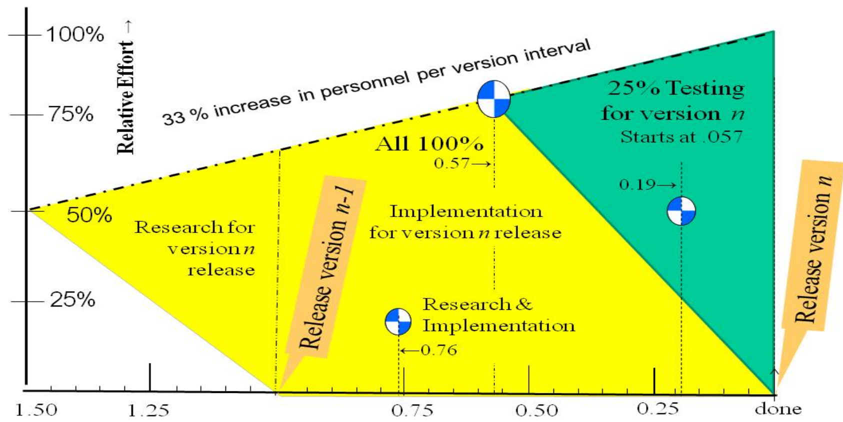
The relative effort expended on these three aspects changes over time. For mature software perfective and adaptive maintenance dominate, since bug fixing decreases. Maintenance activities are known to steadily increase the size of software [HennessyP:90]. This issue, and its effects, has been the topic of a companion paper [Wiederhold:06].

The same three effort components: research, implementation, and testing, that comprise the initial development still have a role. However, during version development it is difficult to identify the research component as a distinct activity. Since the product has been already proven itself, a successor version will require no or little fundamental research. We will hence combine research and implementation in the analyses for version lag.

When successor versions are being developed, much testing must be devoted to assuring that the prior functionality of the product remains intact. A new version of a product must support all the functions of a prior version, and do so in substantially the same manner. Especially for infrastructure software that supports applications, no inconsistencies can be tolerated. All prior test suites are collected and regularly applied. Distinct testing and quality assurance teams will perform such regression testing. Some organizations use daily builds to assure continuing product viability.

5.1 Version Development During Rapid Growth

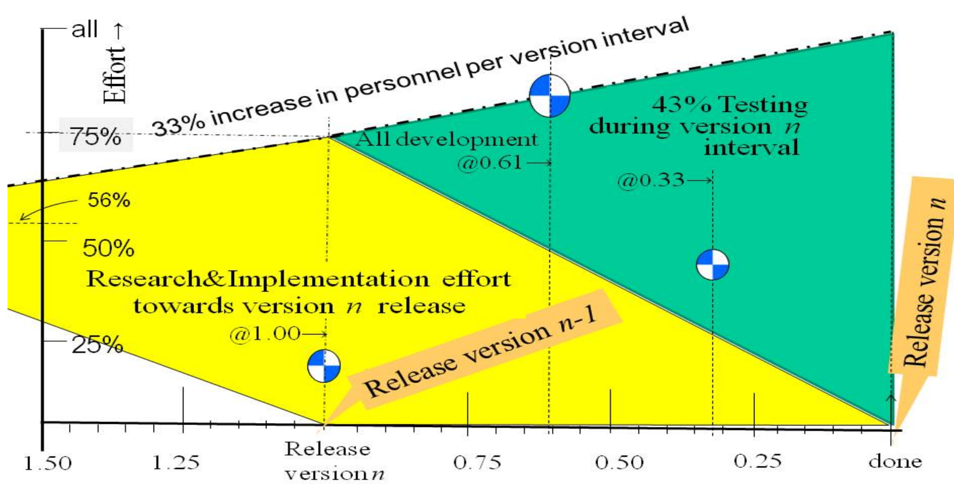
If the initial product promises to be successful, there is much motivation to evolve the product and eliminate any problems to acceptance in the market. Typically, to meet initial deadlines, product features that appeared to be important have been left out. A typical scenario for a company experiencing a high rate of growth, as one would expect if a product is successful, is shown in Figure 11. The reduction of implementation effort when testing starts for the initial product allows the research and implementation teams to commence work on successor versions of the same product prior to the initial product release.



Version development, growth, modest testing Fig.11

The scenario of Figure 11 assumes a 33% increase in staff during each version development period. This rate could be reached given a high annual personnel increase of 25% and versions being released at 18 months intervals. Commencing when testing for the prior version (*n-1*) starts, released research and implementation personnel will start working on the successor version (*n*). For a simple model with 50% testing of the prior product testing this occurs while 0.50 of the prior development cycle still remains; other models of investment growth and testing would allow the reallocation point for the first successor version to be reached somewhat later or earlier. The overall development centroid is now at 0.57 of the version development interval. The centroid for the research and implementation effort is now at 0.76 and the centroid for testing at 0.19 of the version development interval.

For subsequent versions (*n+1*), and this growth rate, even a 25% testing effort will start already at 0.57 during the current version (*n*) development. For successor versions the relative testing requirements increase, since an ever greater fraction of the code comes from prior versions. For versions of mature products, because of concern for reliability and the large amount of existing software, testing will consume about half of the development resources [Maraia:05]. However, we assume that testing cannot start before the prior version is released. In Figure 12 we show a scenario with a substantial testing effort.

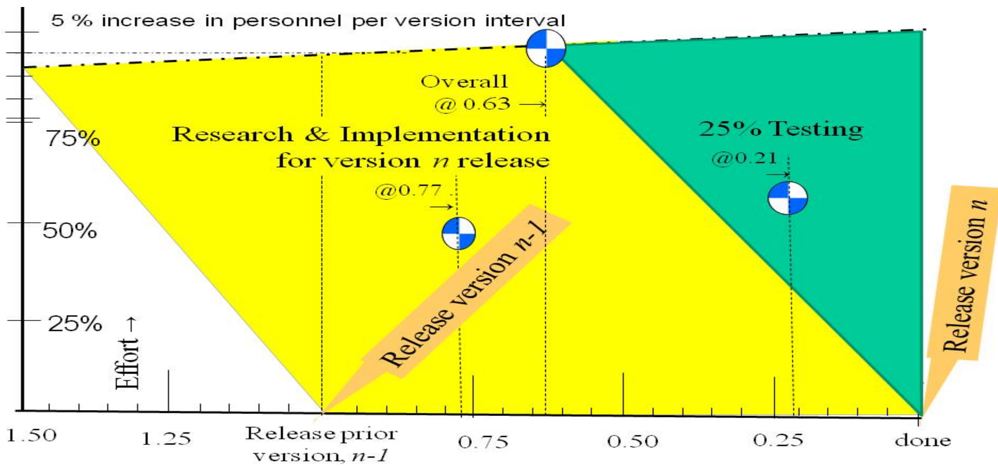


Version development, growth, substantial testing Fig.12

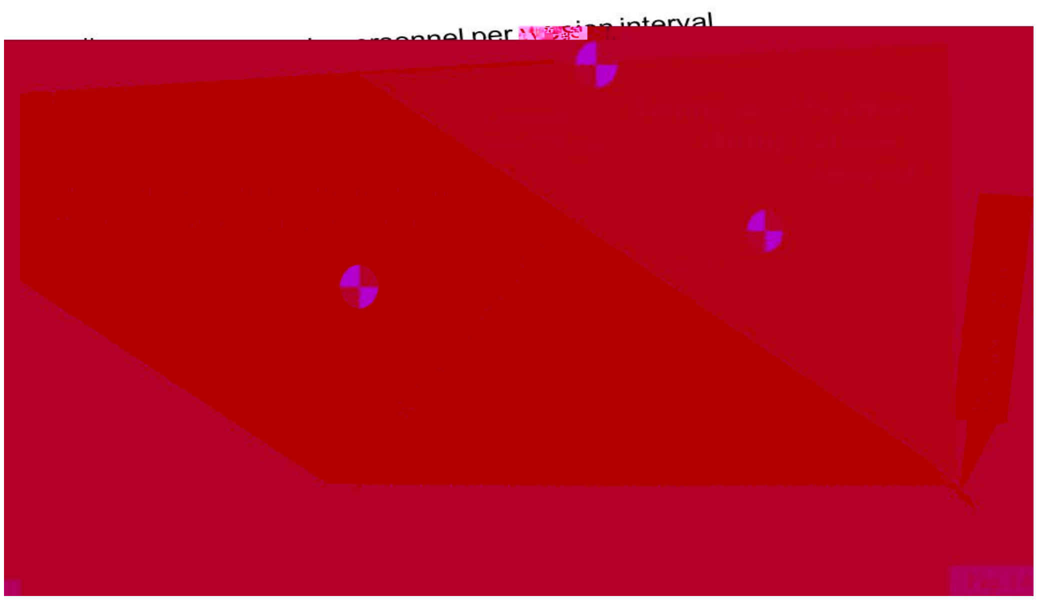
Figure 12 assumes 43% testing starting after the release of the prior version.   The overall effort centroid moves to 61% prior to release to sales. The Research and Implementation centroid alone advances to the start the new version interval. The centroid for testing is now at a 33%.

5.2 Version Development during Mature Growth

Eventually, the exponential growth in personnel effort cannot be sustained. The relative rate of required software changes slows down. Because the body of code has grown, the actual ongoing efforts will be still be substantial. If the growth of personnel becomes less rapid, the centroids will shift to an earlier point in the development interval. We assume now a 5% growth in technical personnel dedicated to version development, appropriate for a mature development cycle. Figures 13 and 14 show the results for the 25% and 50% testing cases.



Version development, mature growth, modest testing Fig.13



The results of all 4 scenarios are tabulated below. They provide a range within which version development lags will fall in practice.

Personnel growth/version	Testing fraction	Total effort centroid	Research & Imple-mentation centroid	Testing Centroid
33%	**25%**	0.57	0.76	0.19
33%	**43%**	0.61	1.00	0.33
5%	**25%**	0.63	0.77	0.21
5%	**35%**	0.77	1.00	0.33

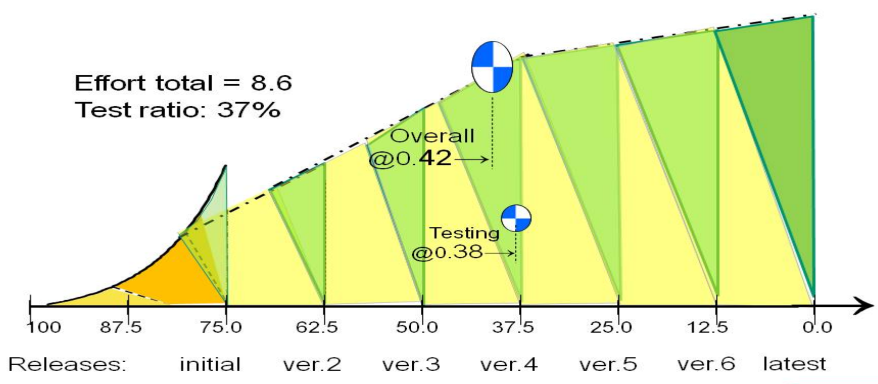
Table 2. Typical version lag parameters and results

6.0 Development summary

We have now considered several types of initial development and subsequent ongoing version development. The worked-out examples represent cases at the low and high range of practical development scenarios. The results vary less than the ranges of input variables, indicating model stability. The results provide general bounds that match experience. We expect that results will stay in same ranges when effort distributions are more complex than the simple linear and small-exponent growth curves used in our examples.

6.1 Multi-version lags

When we look at a product that has gone through multiple versions we must combine the initial development scenarios with the version development scenarios. We show an example in Figure 15. We assume that versions are issued at intervals that are half the length of the initial development period. In the figure we follow a startup initial model by two versioning examples, namely three versions at high growth rates followed by three versions at mature growth rates. In practice the changes will be more gradual. We assume a linear growth of code [Tamai:02]. The relative testing effort increases because the code to be maintained increases steadily.



Multi Version effort and lag Fig.15

By the time the 7th version is released the cumulative effort has been 8.6 times the effort required to develop the first version. Over that time the testing fraction has become 37% from the initial 25%. If we count the costs to develop the incremental versions as investments, the overall lag has become 42%. Indeed, ongoing software development costs are commonly booked as R&D expenses.

6.2 Validation with data

We use here simple models to gain an understanding of the factors that contribute to lag. Within those parameters the total period required for the development is determined by the size and complexity of the product.

The actual shape of the curve can be obtained from records of research, implementation, and test expenses incurred during the creation and ongoing development of a product. Having actual effort data for the research, development, and testing categories allows creation of a diagram showing the actual values for the components shown in the effort figures. From such data a case-specific centroids can be calculated.

Associated with a paper where lag is used for software valuation are spreadsheets that allow alternate values of lag to be inserted [Wiederhold:08S].

7. Lag when Recreating Software.

When an existing product has to be recreated, the issue becomes more complex. One now does not deal only with new product development, where the lag model of Figure 4, 6, 7, 8 or 9 would hold, but with replication of a mature product that has a history covering both the initial lag and likely a number of continuing improvements, as sketched in Figures 11, 12, 13, and 14. One approach to compute the lag needed to develop an equivalent product would be to take the sum of the initial lag and all the subsequent lags that led to the version being assessed.

7. 1 Recreation efforts scope and constraints.

Even when all of the prior software and staff are available, the effects of prior maintenance make prediction of the recreation time needed to produce equivalent but cleaner software has been considered to be well nigh impossible [RugaberS:04]. For situations requiring extremely high reliability, diverse copies of identical functionality may be desired, to serve as backup or provide concurrent validation of results [KellyMY:91]. Such an approach, where only specifications can be shared, implies parallel recreation of software.

Where full compatibility has not been expected, but substantially similar functionality is being provided, recreation has been successful, but still involved much effort and controversy. Lotus added a new interface to Visicalc's spreadsheet functionality, and supported most keystroke features of its predecessor as well, to the extent that it became the subject of a lawsuit. The DISH network's DVR mirrored Tivo's function, but had to reengineer its recreation because of patent misappropriation, and pay substantial damages, following an appeals court ruling in January 2008.

One motivation for recreating software is to move infrastructure software into the open domain. Much knowledge can be shared here, but copyrighted software should not be copied. Efforts by volunteers are often of high quality, but vary in intensity over time [Corbet:07]. Some projects were never completed, as Open Darwin, an operating system for Apple MACs. Open Office, although starting with donated purchased operational source code, took five years to become functionally competitive, although is still not fully compatible with its progenitor, Microsoft Office. Understanding the effort needed, and the expected lag can help in planning open source projects, and perhaps limiting their ambitions.

Software of modest size has been recreated effectively without direct access to prior code. The most well-known instance of generating perfect functional copies have dealt with modest code sizes, for instance the BIOS developed by Phoenix in 1983 for compatible PCs required less than 32K of code for original the CompaQ PC clone [Schwartz:01]. Another was VTech's successful cloning of the Apple II ROMs for their Laser 128 PC. Lags were still substantial.

7.2 The problem and factors affecting software re-creation

The effort needed to recreate a product depends on the available information and knowledge. In each case we assume that the original code for the original product is not available or not useful for outright copying. We distinguish internal and external information, and in each case must consider code, documentation, and expertise.

1. **Internal information**
   - 1.1. Is the original code available for inspection? In 1989 Fujitsu paid IBM $51M to read, but not copy portions of OS/360 to help them recreate an updated version of the prior version. Note that Fujitsu prior to that date had used and provided that prior version to its customers [Jussawalla:92].
   - 1.2. Is internal documentation available? Typically the same restrictions apply as to original code. Further internal documentation, especially after many updates, is also notably untrustworthy [Spolsky:04].
   - 1.3. Are some of the original implementers available, and can they share their knowledge without violating employment covenants?
2. **External information**
   - 2.1. Is all of the binary code available and executable? For marketed products that is typically true.
   - 2.2. Is the external documentation available? Again, typically true for marketed products, but such documentation may not describe all features. Undocumented features may support testing, demonstration setups, and performance enhancements, and perhaps features focused on specific major customers.
   - 2.3. Are experienced users available? Having experienced users can overcome some of the puzzles encountered when attempting to replicate a product. Experience is valuable even for executing the original binary programs or reading its external documentation.

In the cases cited in the introduction, it is often a contention how much information from the source was available to the replication effort. In order to focus, we will assume that no internal information is available, but that all external information is available.

7.3 Arms-length software re-creation

At times there is a requirement for a company to operate "at arms length"; disallowing any use of internal information. Without access to internal code, documents, and knowledge, a re-creation attempt requires reverse engineering. In a formal setting the staff performing the recreation is isolated in a clean room and can receive only results from external testing [Schwartz:01]. We also assume that the competence of the staff working on recreating the software matches the competence of the original software authors.

The problem to be addressed in our context is: what is a fair estimate of lag, the time that a company operating at arms-length (COAL) from the current supplier of the software would have to spend in order to have its own equivalent software?

The replication effort typically has to create software that was created originally and subsequently improved through a number of versions. The effort needed for re-creation could then appear to be the sum of all the efforts represented by the initial and the subsequent version lags. If the personnel quality and number available is similar, then the re-creation lag would be equal to sum of all those lags. However, a number of factors would alter that estimate:

1. An aggressive COAL would be able to put more staff on the re-creation task in order to reduce lag than was available during the original development. However, there are limits to that strategy. The initial design team has to be constrained to a group that can communicate frequently and easily, typically less than a dozen people. The staff can grow as soon as an initial design document has been produced and accepted. But even then there are limits to personnel growth. In Section 4.4 we compared lag times between start-ups and mature companies. For initial development the ratio of 1.58/0.63 =2.5 shown in Figure 10 would be the maximal reduction in lag between a startup and a mature company with ready and experienced resources. However, as long as the effort is constant the cost is unlikely to differ to any great extent.

   Since the COAL is unlikely to be a startup, nor a mature company with many well-organized resources, we consider that use of the simple model for the re-creation effort provides the most reasonable compromise. For a specific case this assumption should be verified.

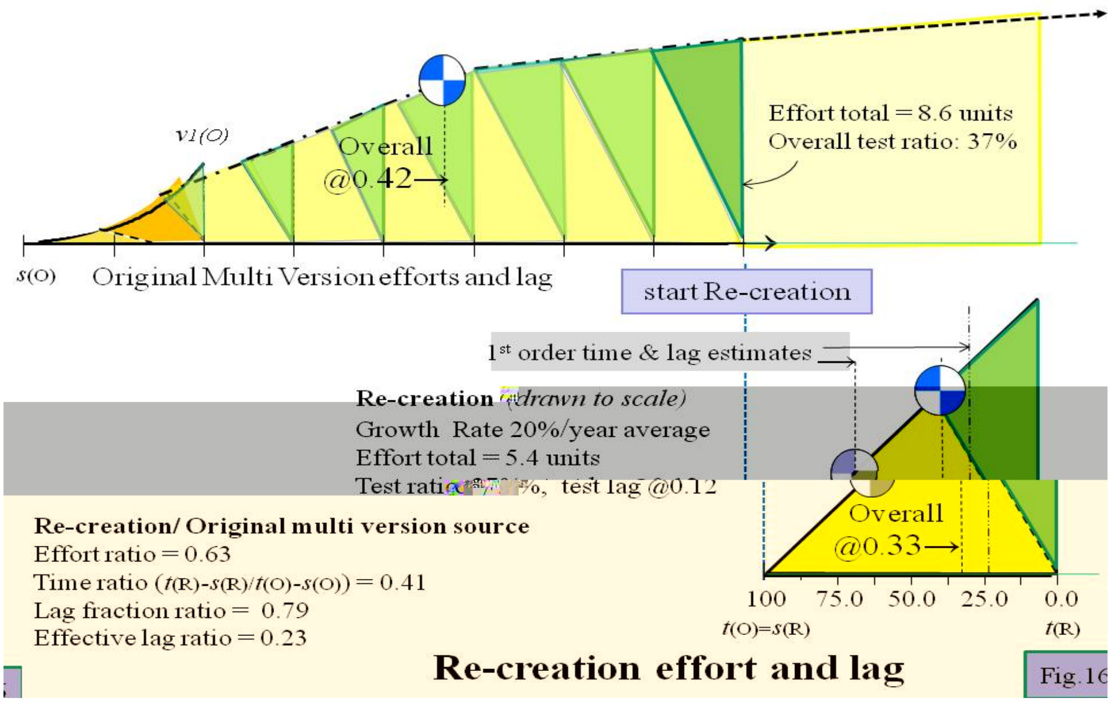
2. The effort required to recreate a product should be less than the effort spent in original creation. Less research will be needed since important questions have been resolved. Some work performed for one original version may be superseded in a successor version. In a mature product, some parts will likely have been rewritten. For instance, the standards that must be complied with are well known to a COAL, so that the initial design can accommodate them all.

   Our earlier work posited a 5% annual deletion rate of code, applied to the body of code existing when the version update was initiated [Wiederhold:06]. Given an 18 month version interval, and expected version growth, we can estimate an effort reduction under the assumption that no code is being superseded during the re-creation effort. For versions 2 through 7 the amount of superseded code becomes 4%, 9%, 14%, 20%, 27%, and 33% [Wiederhold:08S]. Aggregating these savings, and assuming that in the re-creation process no code effort is wasted, leads to an effort reduction amounting to 63%, and, for the simple model, a time reduction to 41%.

3. The rate of bugs, and hence the amount of required corrections tends to be proportional to code size. The cost of fixing a bug tends to increase with code size. The amount of testing for a monolithic re-creation of the code being worked on will be large. We should assume that substantial testing is needed for a truly compatible product.
   For our estimate we use a testing ratio equal to the aggregated testing experienced during the original product development, as shown in Figure 15, namely 37%.

4. Perfective maintenance is based on feedback from the field. While a COAL will have information about details of the current original product, that original can also be improved during the time the COAL recreates the product. The COAL would not have the database to drive effective perfective maintenance. The initial recreated product will hence lag behind improvements made in the original product, and likely require at least one more iteration of effort.
   We ignore the delay needed to catch up and assume that the initial recreated product is adequate.

If we take these 4 points into consideration as indicated, we can model the re-creation effort fairly, as shown in Figure 16.



Re-creation effort and lag
Fig.16

The total re-creation effort is derived from the sum of the initial and all version efforts, adjusted by omitting any effort due to code that was superseded in any version effort. We assume a high, but linear rate of staff growth, and that the new staff has equal competence and productivity as the staff that created the original product and its versions. The staff rises rapidly to a greater headcount. The interval for re-creation is now only 0.41 of the total development time of the original product to version 7, or 1.65 the time needed to create the initial original version [WiederholdS:08S]. Since the development interval is shorter, the lag at the 0.33% centroid is now also much less.

A first-order estimate for a re-creation lag can be obtained by taking the sum of the initial effective lag and the average effective version lag for the development of the original software. This estimate does not require an estimation of the length of the re-creation effort, but assumes implicitly that the replication takes as long as the creation of the first version of the product. Those points, at 0.74 for the re-creation interval of the life and 0.51 for the lag within that period are also marked for the sample shown in Figure 16. This estimate can be corrected for the expected length of the re-creation interval once the capabilities, i.e., the maturity of the organization performing the recreation are known.

The length of the re-creation interval determines the delay before income can be realized. The lag is only a metric of the investment pattern. In the case shown in Figure 16 the re-created product is ready for sales in 0.41 of the time that the original software development required to get to version 7.0, but there is no income at all until the re-created product is fully ready. Although the investment lag fraction is less, the economic benefit of the investment is delayed to a much greater extent, as is the risk, because no feedback from the market can occur until the product is actually sold. To assess these effects, one would need to make assumptions that go beyond scope of the engineering business assumptions we made throughout this paper.

8. Summary and Refinements

This paper provides methods for estimation of investment lags for a variety of typical conditions. In order to obtain reasonable bounds we analyzed a hypothetical software product development in a variety of settings. For the initial product we considered three types of development setting:

- Section 4.1 A steadily growing software development group, the simple model
- Section 4.3 A startup, with limited initial resources
- Section 4.4 An existing mature company which can rapidly marshal resources

We then considered similarly the development of successor versions of a successful product. Here two scenarios were considered

- Section 5.1 A rapidly growing company, marketing and improving a novel product
- Section 5.2 A mature company, marketing and improving a more stable product

In each of these two cases we considered testing of prior versions at a modest level, allowing starting a release of implementors midway during the prior version development cycle, and at a substantial level, allowing the release of implementors throughout the prior version development cycle.

In order to get insights into the overall investment patterns, we combined an initial startup product development with 6 successive version releases of a successful product. We then assessed the effort needed to recreate such a product. Re-creation of software is complex, and has been rarely quantitatively analyzed. In order to demonstrate a likely case we ignored both substantial negative and positive factors. Overall, the re-creation result appears to be optimistic, but the available data are so sparse that we cannot validate the re-creation model with actual experiences.

For each scenario we also considered typical relative efforts for research, implementation, and testing. Separating these components allows refinement of investment planning, since distinct personnel will be involved, likely at different rates of reimbursement and incentives. We do not try to assess here the effects of differing pay rates on lag, but have provided enough information to allow such refinements to be made. If quality assurance is performed off-shore, the effect can be substantial.

Section 7 of this paper deals with this issue in more depth, but the quantification requires many assumptions.

Getting a good handle on ongoing development and maintenance costs that do not increase functionality in novel ways is difficult. Those costs are typically booked together with new research projects as research-and-development expenses. FASB guidance requires expensing of ongoing costs, but does not indicate how [FASB:85]. Since ongoing improvement is an essential characteristic of software, and is necessary to keep software marketable, it might be best to categorize ongoing expenses as a subcategory of `Cost-Of-Goods-Sold' (COGS). Marketing costs for current products are also typically accounted as part of COGS or as General and Administrative (G&A) expenses. It would also be useful to have a subcategory for Marketing and Sales costs, since they are investments distinct from overhead. Not considering software maintenance costs as R&D would also reduce the irrational gross profit margins that are now reported by software companies, but no such changes are on the horizon, even while the problems in dealing with the financial metrics of intangible development are being debated [Lev:01].

Acknowledgements

The importance of lag became evident in discussions associated with the application of software valuation methods described in [Wiederhold:06]. Especially the issues of lag for recreation deserved attention. I received valuable feedback for this paper from Joaquin Miller and Shirley Tessler. I look forward to further inputs on the issues raised.

References

[AbdelHamid:93] T.K. Abdel-Hamid: "Adapting, correcting, and perfecting software estimates: a maintenance metaphor"; *IEEE Computer*, Vol.26 no.3, March 1993, pp.20-29.

[AmblerS:96] Tim Ambler and Chris Styles: "Brand development versus new product development: towards a process model of extension decisions"; *Marketing Intelligence & Planning;* MCB UP Ltd, Vol.14 No. 7, pp.10–19, 1996.

[BarIlanS:96] Avner Bar-Ilan and William C. Strange: "Investment Lags"; *American Economic Review,* American Economic Association, Vol. 86 No.3,1996, pp. 610-622.

[BarIlanS:98] Avner Bar-Ilan and William C. Strange: "A model of sequential investment"; *Journal of Economic Dynamics and Control*, Elsevier, vol. 22(3), pp.437-463, March 1998.

[Bernstein:03] Philip Bernstein: Remark at NLM/NSF planning meeting, Bethesda, MD, 3 Feb 2003.

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

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

[Corbet:07] Jonathan Corbet: Who wrote 2.6.20?; http://LWN.net/articles/222773, 20 Feb. 2007.

[Damodaran:05] Aswath Damodaran: The Promise and Peril of Real Options; Stern School of Business, 2005.

[Desmond:07] John P. Desmond: "Applications Go WorldWide, Survey of 500 Software Companies"; *Software magazine*, Oct. 2007, p.16-57.

[FASB:85] Financial Accounting Standards Board: Accounting for the Costs of Computer Software to be Sold, Leased, or Otherwise Marketed - FAS 86; 1985.

[Frank:05] Steven J. Frank: "Original Out, Risk In"; *IEEE Spectrum,* April 2005. pp. 60,62.

[Glass:03] Robert L. Glass: *Facts and Fallacies of Software Engineering*; Addison Wesley, 2003.

[Graham:1994] Dorothy Graham: "Testing"; in J.J. Marcianak: *Encyclopedia of Software Engineering*; Wiley, 1994, p.1330-1354.

[HennessyP:90] John Hennessy and David Patterson: *Computer Architecture*; Morgan Kaufman, 1990 (3rd Edition 2002).

[IEEE:02] IEEE Standard Glossary of Software Engineering Terminology; 610.12-1990, (R2002), revised 2002. .

[Jussawalla:92] Meheroo Jussawalla: *The Economics of Intellectual Property in a World Without Frontiers;* Praeger/Greenwood, 1993, p.113.

[Klepper:96] Steven Klepper: "Entry, Exit, Growth, and Innovation over the Product Life Cycle"; The American Economic Review, Vol. 86, No. 3 (Jun., 1996), pp. 562-583.

[KruchtenOS:06] P. Kruchten, H. Obbink, and J. Stafford: "The Past, Present, and Future for Software Architecture"; *IEEE Software*, Vol.23 No.2, pp.22- 30.

[Lammers:86] Susan Lammers: *Programmers at Wor*k; Microsoft press, 1986.

[Lev:01] Baruch Lev: Intangibles, *Management, Measurement and Reporting*, with comments by conference participants; Brookings Institution Press, 2001.

[LevS:96] Baruch Lev and Theodore Sougiannis: "The Capitalization. Amortization, and Value-Relevance of R&D"; *Journal of Accounting and Economics*, 1996, pp.107-128.

[Maraia:05] Vincent Maraia: *The Build Master, Microsoft's Software Configuration Management Best Practices*; Addison-Wesley Microsoft Technology, 2005.

[Markusen:90] James R. Markusen: First Mover Advantages, Blockaded Entry, And the Economics of Uneven Development; NBER Working Paper No. 3284*
Issued in March 1990. RB case.

[McConnellB:05] Campbell R. McConnell and Stanley L. Brue: *Economics: Principles, Problems, and Policies;* McGrawHill, 2005. Lag discussed on p.223, 494

[McKinsey:03] McKinsey: "Fighting Complexity In IT"; McKinsey Quarterly, April 2003 at http://www.forbes.com/technology/2003/03/04/cx_0304mckinsey.html.

[MulfordR:06] Charles W. Mulford and Jack Roberts: Capitalization of Software Development Costs: A Survey of Accounting Practices in the Software Industry; College of Management, Financial Analysis Lab., Georgia Tech., May, 2006.

[Pfleeger:01] Shari Lawrence Pfleeger: Software *Engineering, Theory and Practice*, 2nd ed; Prentice-Hall, 2001.

[PMI:04] Project Management Institute: *A Guide to the Project Management Body of Knowledge;* IEEE Std 1490-2003, http://standards.ieee.org/reading/ieee/std_public/description/se/1490-2003_desc.html, Oct. 2004.

[PrattRS:00] Shannon Sharon Pratt, Robert F. Reilly, and Robert P. Schweihs: *Valuing a business: The Analysis and Appraisal of Closely Held Companies,* 4th Edition; McGraw-Hill, 2000.

[RavenscraftS:82] D. Ravenscraft and F,M. Sherer: "The Lag structure of Returns to Research and Development"; *Applied Economics*, Vol. 14, 1982, p.603-620.

[RugaberS:04] S. Rugaber and K. Stirewalt: "Model-driven reverse engineering"; *Software IEEE*, Vol.21 No. 4, July-Aug. 2004, pp.45–53.

[Schwartz:01] Mathew Schwartz: "Reverse Engineering"; *ComputerWorld*, 12 Nov. 2001; http://www.startupgallery.org/gallery/story.php?ii=57.

[Sodal:01] Sigbjorn Sodal: Entry, Exit And Scrapping Decisions With Investment Lags: A Series Of Investment Models Based On A New Approach; Department of Economics, UC Santa Barbara in its series University of California at Santa Barbara, Economics Working Paper Series, No.1058, March 2001.

[Spolsky:04] Joel Spolsky: *Joel on Software*; Apress, 2004; based in part on http://www.joelonsoftware.com, 2001.

[Tamai:02] Tetsuo Tamai: "Process of Software Evolution"; *First International Symposium on Cyberworlds*, Tokyo, Nov. 2002, p.8-15.

[WiederholdE:71] Gio Wiederhold and John Ehrman: "An Inferred Syntax and Semantics of PL/S"; *Proceedings of the ACM Conference on System Implementation Languages*, SIGPLAN Notices, Vol.6 No.9, October 1971, pages 111-121.

[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:06] Gio Wiederhold: "What is Your Software Worth?"; *Comm. ACM*, Vol. 49 No. 9, Sep. 2006, p.65-75. Full paper at http://infolab.stanford.edu/pub/gio/inprogress.html#worth

[Wiederhold:08S] Gio Wiederhold: Spreadsheets Documenting Lag Estimation; 2008, http://infolab.stanford.edu/pub/gio/inprogress.html#lag (to come)

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