

Turn Around!

A brief guide to starting, growing or turning around your software and Internet company.



*Version 31. March 26, 2001. First version released January 15th, 2001.
Text and photos copyright Alistair Davidson, 2001. All rights reserved.*

Contact information:

*Alistair Davidson,
Eclicktick Corporation,
1560 Sand Hill Road, Suite 304,
Palo Alto, CA 94304.
Phone 650-322-8880/415-225-8610 cell
E-mail: alistair@eclicktick.com*

Copies of this document are available at <http://www.eclicktick.com>

Turn Around!

Turn Around!	1
Introduction.....	7
Chapter 1: The Key Guidelines	8
Build Great Software!	8
Hire Great Programmers. Fire Bad Programmers.	8
Training Does Make A Difference	8
Software Solutions Need to Be Engineered	9
Anticipate That Customers Change	9
Don't Overestimate Customers	9
Get People Up and Running Quickly	10
Market Prioritization Should Be Based Upon Sales Cycles	10
Market Research	10
Managing Technical Constraints Is Always An Issue	11
Sell Benefits, Not Features	11
Build a Brand That Leapfrogs RFI/RFQ Evaluation Processes	12
Don't Give Software Away For Free	13
Product Management is a Critical Activity	13
Constraint Management	14
Productizing While Bootstrapping	14
Productizing With Venture Capital	15
Do Usability Testing	15
Launching Products Takes Money – More Than It Took to Develop the Product	15
Software Margins Aren't as High As They Appear	16
Why the Sales Cycle is So Important	16
Sales Cost	16
Sales Forecasting	17
Cash Flow Forecasting	18
Support Costs Matter	18
Timing Is Critical	18

Turn Around!

Balance Installed Base Marketing vs. Customer Driven Requests	19
Talk and Listen Closely to Customers	19
Working Capital and Relationship Profitability.....	19
Get Customers.....	20
Build a Business With Negative Working Capital Requirements	20
You Need Reference Sites.....	20
Pricing Should Be Based Upon Economic Value to the Customer.....	20
Pricing vs. the Competition.....	21
Specialist Software and Tool Firms Have a Good Future.....	21
Time-Based Competition.....	22
Do Things in the Right Order	22
Business Models	22
Chapter 2: Building a Compelling Software Capability	24
Process vs. Output Benefits.....	26
Assessing a Compelling Software/Service Product.....	28
Product Assessment Rankings	28
Chapter 3: Common Problems in Software Teams and How to Resolve Them.....	29
The Interface Problem.....	29
An Example of a Failed Development Process	29
Long Development Cycles	30
Relying Upon the CTO/Architect to Specify the Product	30
Documenting in Parallel.....	31
A Key Assumption About Software Development.....	31
Tying the Business Model to the Technical Development Strategy	32
The Back of the Envelope Calculation.....	32
Chapter 4: Marketing and Sales.....	34
Hiring Good Marketing and Sales People	34
Managing the Vision; Listening to Customers.....	34
Pioneers vs. Settlers	34
The Software Sales Presentation	35
The Lemming Industry.....	36
Public Relations.....	37
Software Evangelism	37
Marketing Software	37

Turn Around!

Some Marketing Guidelines	38
Test Your Results	39
Trade Shows	39
Literature Development	40
Chapter 5: Promoting the Web Site	41
Chapter 6: Defining Your Competition	43
The Economics of Different Solution Approaches.....	44
Becoming a Standard	46
Semi-Open vs. Open	46
Pricing Software on the Internet	47
Chapter 7: Improving Sales Performance	48
There is a limited window in which you can sell your high tech product	49
Segmentation choices affect the length of your sales cycle. Testing sales cycles and focusing upon segments with rapid sales cycles dramatically improves performance	49
The high tech sales cycle typically has six components.	51
There is a clear trade-off between short-term usage goals and relationship profitability	52
Rapid evolution of high tech products improves product performance and customer satisfaction.	53
Rapid evolution requires improved performance measurement systems.	54
Product development is about both product and services development	54
Highly leveraged business models are not restricted to technology-based strategies.	55
Total Margin	56
CRM requirements are rarely planned for and are a significant component of the selling process.	56
Distribution and strategic alliance strategies are a source of costs that are typically underestimated by companies.	57
Measurement of the impact of price, feature sets and different value propositions is rarely attempted.	57
Market research can help far more than most firms realize.	58
Action Steps	58
Chapter 8: Managing the Team	59
Rule 1: Protect the Developers	59
Rule 2: Don't Have Developers Do Marketing's Job	59

Turn Around!

Rule 3: Talk the Talk and Walk the Walk	60
Rule 4: Measurement Matters	60
Rule 5: Project Management Matters.....	61
Rule 6: Sequencing Matters	62
Rule 7: Turnover is Expensive.....	62
Rule 8: Treat Customers as You Would Want to Be Treated	63
Dealing with Performance Problems in the Team.....	63
Values and Philosophy.....	63
Project allocation problems.....	63
Programming skills and aptitudes.....	64
Communications skills.....	64
Chapter 9: The Internet	65
The Economics of Software and the Internet	66
Noise on the Internet.....	67
Software On the Internet.....	67
The Functionality Dilemma	68
The Grammar of Software and the Problem of Procedure	69
Chapter 10: Humility and Other Matters of Judgment	72
Put the Customer First	72
Know Your Negotiating Power	73
A Useful Sales Framework.....	74
Chapter 11: Managing For Value	76
Measuring Free Cash Flow	76
Defining a Sustainable Business	77
Dealing with Missing Information.....	77
Modeling Value Creation	77
Managing and Measuring Initiatives	79
Pareto Portfolio View of Value Creation	80
Chapter 12: Financing the Software Business	81
The Financing Decision	82
The Magic of Success	85
If You Are In Trouble.....	85
Chapter 14: Is Your Business a Dog?.....	87
Rating Your Business	87
Evaluation of the Ratings.....	92

Turn Around!

Chapter 15: Turning Around Your Internet Software Business - a guide for crisis handling	93
Admit that you have a problem	93
Involve the Management Team	93
Classify Your Activities	94
Strategies for improving your profitability	94
Develop a Profile of Your Strengths and Weaknesses.....	95
Rank Your Customers.....	95
Charge More for Your Services.....	96
Approach Different Types of Investors.....	96
Change Your Message	96
Stop Developing New Stuff.....	97
Motivate the Team	97
Keep Your Eye On the Ball.....	97
Don't Flog A Dead Horse	98
Decision Making Under Stress.....	99
Chapter 16: The Strategic Perspective.....	100
Gap Analysis	100
Appendix: More Useful Reading.....	101
The author: Alistair Davidson.....	102

Turn Around!

Introduction

Software is an easy business to get into, but it is a tough business to make money at.

This document is an attempt to lay out a philosophy of what a software or Internet company or, in general, a high tech firm, needs to do to make money. It is based upon my experience as founder of a software company, as both a strategic and IT consultant, as CEO of an Internet company, and advisor to Internet companies. The ideas borrow widely from the many talented people I have worked with and assume that a key part of your value chain involves creating value through software.

Every company is different. *Every company needs its own strategic and operating plan, that the management team has developed together*, but deviating from the precepts in this document has often proven costly to companies that I have observed.

The focus of this document is the startup or recently formed software or Internet company that is attempting to build a profitable software-based business, or a company that has run into growth and financing problems. In either case, change is probably a prerequisite for growth or recovery. Carrying on without change will mean going off a cliff.

For the majority of software companies providing applications and tools primarily in B2B markets and Internet companies where software and information management is a core activity, I hope you will find my comments useful. Chapter 11, *Managing for Value* is particularly appropriate for firms with multiple products and larger size.

Turn Around! is based upon our firm's observation that most companies with a high software and informational component make a consistent pattern of mistakes. These mistakes, if avoided, can help companies spend less on R&D, have more successful launches, and achieve higher levels of profitability and market share.

Readers with narrow backgrounds and perspectives may find *Turn Around!* frustrating. We grapple with issues of staffing, software strategy, product management, marketing and launch strategy, sales strategy, organization learning, the economics of the Internet, planning approaches, measurement and performance issues, how to create value.

But success as CEO and success for a company are based upon integrating all these disparate views of a business. I have written the book, as a result, as an overview and a set of reminders. One reader actually said he needed to read this material every month to remind him of best practices. I hope you will find the material just as useful.

As this book is a continually evolving work, I solicit input from readers and am happy to incorporate suggestions, anecdotes or references (with credit or anonymously as you prefer). I can be reached at alistair@eclicktack.com.

Turn Around!

Chapter 1: The Key Guidelines

Build Great Software!

At its simplest, a software company needs to build great software, software that its customers love and come to depend on. In quality management terms, software should *delight* its users.

The research on new product development proves very clearly that companies that have new products with customer perceivable, high and differentiated value are both (1) more likely to succeed and (2) achieve higher market share than companies with me-too products.

Hire Great Programmers. Fire Bad Programmers.

The recipe for building great software is to have great people who have passion for their software. Software *never* emerges by accident. There are always one or more people who care about the product. No good software was every created without a passionate architect or team.

Mediocre and bad programmers merely waste the time of the good people.

Surprisingly, most customer organizations don't do a very good job of firing bad programmers. The reason is that they have difficulty measuring performance. Another reason is perceptual. Most managers don't actually understand the wide range of performance that exists between brilliant and bad programmers.

In my own experience, I have led companies to achieve a 20X improvement in productivity. But it took the hiring of great developers, great effort, a technical vision, risk taking and complete change in both programming paradigms, management processes, and also the tools used. Without the support of senior management, this type of performance improvement will not occur.

Fundamentally, most organizations have never achieved a large improvement in software production. So they do not believe it is possible. Great software companies believe great software and performance improvement are possible.

Training Does Make A Difference

Most companies don't train their staff. They should. Learning how to use a package is not the same as knowing how to use it well. Training has a very high return on investment and perhaps just as importantly, it begins the process of becoming an *employer of choice*. Companies that achieve the status of *employer of choice* are like sports teams with great draft picks. They win.

Turn Around!

Software Solutions Need to Be Engineered

When I first started off building software, one of the lessons that I had to learn was the need to engineer software. Engineering software has five basic elements:

1. Estimating project size and complexity.
2. Decomposing projects into doable chunks.
3. Identifying components that can be contracted out or purchased.
4. Monitoring performance and revising schedules and resourcing.
5. Testing.

I have rarely run into already initiated projects where failing to scope and plan the project did not bring enormous problems. But even if you scope the project, decomposition of projects can also be done badly. The ideal decomposition allows pieces of the project to get used early on and tested early on.

And is widely understood based on the famous book by Frederick Brooks, *The Mythical Man Month*, project teams need to be kept small. The magic number for a team is around seven people. Adding more people to the team reduces productivity because the amount of work goes down as the amount of communicating time goes up.

Over the past fifteen years, there has been huge debate about how to increase software productivity. In general, the best approach is to avoid programming. Selecting development environments with rich functionality and licensing third party components and technology are both important time and effort savers. So is adoption of standards that save reinventing interfaces to other software. With the advent of the Internet, this logic can now be extended to integrating services from ASP vendors.

In fact, there are so many services on the Internet that frequently, the evaluation process is actually one of the most difficult parts of software development and strategy today.

Anticipate That Customers Change

What makes software difficult is that the user is always a changing a target, so software needs to be designed to support the level of skills and changing needs of a customer. Software, like the military, changes people. It transforms the customer.

Don't Overestimate Customers

Good software builders tend to be exceptional bright and effective people. Unfortunately, the act of building software creates a problem – the person who has built it can no longer understand the mind-set of the person using the software.

That's why marketers got invented. They are the people who have not built the software and they stand between the developers and the customers. They figure out what the customers and prospects believe to be true about their needs, what installed equipment

Turn Around!

and software they have, and come up with a marketing message that is pegged to where the customer is, not where they should be.

Fundamentally, customers are distracted people. They don't understand what your software does. They are not very good at learning it. They are pleased by early successes. So all software must be designed to give the software early successes.

Get People Up and Running Quickly

The most powerful sales tool is success. So it is not enough to sell to a customer. What is important is having the customer create something that works. Repeat business is the foundation of any good business and is particularly important with software tools and applications. So the sales cycle has three parts:

- Time to close
- Time to competence
- Time to productivity

Once a customer is up and running, he needs to be able to justify to the economic decision makers in the organization why it has been an effective purchase. Sales tools need to help the customer buy the product, become competent and demonstrate that success has occurred.

Market Prioritization Should Be Based Upon Sales Cycles

The best customers are those that buy immediately. Most software businesses die because their sales cycle is too costly.

There are three major solutions to long sales cycles:

1. Sell to different people with shorter sales cycles.
2. Invest in marketing to shorten the sales cycle.
3. Simplify and focus your benefit message.

Market Research

The cheapest market research for any startup is always from the magazine serving its industry. All magazines have advertising kits. More successful magazines survey their readers and in many cases, they provide wonderful data to their advertisers. It may be self-serving but it is a good place to start.

Trade associations are also inexpensive sources of data. And it is amazing how much data is available from government organizations as well.

Sales people are also an excellent source of market information if they are disciplined enough to collect and profile prospect information.

Turn Around!

In my experience, most startup software companies do too little research, too late in the development process, and generally don't believe the data they receive.

Involving marketing and research professionals early in the development of the project tends to lead to higher success, greater market share and higher customer satisfaction.

Managing Technical Constraints Is Always An Issue

There are never enough good technical people in an organization.

So, organizations have to be clever in their allocation of people and tasks.

Some key insights that I have learned over the years from my chief technical officers include the following:

1. Recognize that there is a difference between the role of Chief Technical Officer, the Architect and the VP Engineering. The Chief Technical Officer is like the CEO. He represents the technical team to the world and can take a longer term view on the company, its staffing, its training approach, its measurement, its tool selection. The Architect is the artist – he defines the framework from which all the developers must work. The VP Engineering is concerned with project management, and managing the task of resourcing the project, from advertising for staff to hiring and meeting deadlines across multiple projects. It is almost impossible to play all three roles in a live functioning business of any size. If there are constraints on hiring, then the person playing multiple roles needs to be very careful in his time allocation, but it won't work for long.
2. Recognize that there is a difference between Product Marketing and Technical Product Management. Occasionally, you will find a person that can do both. But it is rare because a good marketer is focused upon customers. A good Technical Product Managers is focused upon understanding the functionality of the product and managing its evolution based upon the marketing objectives and feedback.
3. A product specification is the single most important weapon is bringing a product to market on time and on budget.
4. Large product specifications don't generally work. They are too grandiose and cannot be implemented in the time available.
5. Projects should be built quickly so that visible deliverables are reviewable every two to three weeks.

Sell Benefits, Not Features

When you have been building software for a long time, both managers and developers become knowledgeable about the software tools or application can do.

The challenge is always to translate the capabilities of the software into benefits. Now many marketers will put together elaborate tables of features, advantages and benefits.

Turn Around!

Those are useful exercises, but people don't typically buy based upon features. They buy based upon emotions and perceived brand value, or based upon a business case which clearly demonstrate the benefits of the software.

The challenge therefore, is to communicate one overarching benefit that will make them want to buy. Everything else is a supporting benefit not the primary one that will be the focus of marketing.

Selling benefits sounds easy. But it is amazing how even good marketing people can get trapped in the features view of the world. Customers don't understand your product and they don't care about the things that later, as users, they will care about. They will buy based upon the benefits *only* if they are translated successfully into the language and vocabulary of their industry or situation.

Build a Brand That Leapfrogs RFI/RFQ Evaluation Processes

The ideal branding position is one where the leadership of the software is so immense that evaluation of competitors occurs rarely or is a formality.

Few companies achieve this dominance, but it is a good objective. Examples include:

1. MS-Office
2. MS-Windows
3. Oracle
4. AutoCAD
5. Palm Pilot Operating System

In spite of the fact that you can obtain a free equivalent product to MS-Office, the Star Office from Sun, few actually use the rather high quality offering from Sun. Few even think to consider Lotus' offering any more.

Dominance typically results from seven factors;

1. High market share
2. Widespread adoption
3. Excellent fit between functionality and need
4. Generally, more functionality than is actually used
5. Simplicity of operation
6. Widespread support and expertise
7. Media support via magazines

Price is not actually a key issue. The reality is that the real cost of software is not the tool or the application, but rather the cost of maintaining and supporting it, or the business benefits created by it.

Turn Around!

Don't Give Software Away For Free

Many people think that giving away software for free is a good idea. Proponents of this idea have various rationales:

1. **Shareware:** software is priced so low that no marketing or sales effort can be afforded. Giving it away is an attempt to get trial usage. Closing the sale is based upon honor or making available incremental functionality such as no advertising, support, additional features and upgrades.
2. **Installed base upgrade model:** hook them on the product and get them on the next generation. This traditional Internet model is extremely hard to make profitable.
3. **Advertise to users:** the software is merely a device to create a subscription base. The parallel here is with *controlled circulation* magazines (i.e. free magazines/newspapers targeting a particular audience region or specialty audience). Sometimes it is a difficult strategy, but one that can work. It only tends to work if you offer advertisers a group that they want to reach and normally have difficulty reaching cost-effectively. One problem with this model is that requires being good at both software and also selling advertising.

The reality is that many companies give away software and never succeed in migrating their customers to a paying status. You really have to have a compelling upgrade to persuade customers to pay, once they have become accustomed to the free product.

There are few shareware models that have actually made a lot of money. Frankly, the problem is that the barriers to entry are typically very low and differentiation is difficult.

But there are times in a company when giving software away for free does work. In my experience, one of the best examples was a former distributor. He had software that he gave away for free as part of a management seminar. But he charged for the management seminar. And he always made money. His customers always left the seminar knowing the theory behind the tool and how to run the tool.

Other distributors sold the software and only occasionally offered a seminar. They never made any money.

Product Management is a Critical Activity

Software product marketing and technical product management is normally the most likely point of failure in a software company. The reason is that product management is where three sets of issues intersect:

1. The customer and marketing requirements;
2. The product architecture and the supportable features; and,
3. Resourcing.

It takes a special kind of person to understand all three and be able to coordinate experts in all three roles. Companies basically have two choices. They can appoint a strong

Turn Around!

product manager who is in effect the CEO for the product. This approach is a requirement in a multi-product company. Or they can appoint a weak product manager whose role is less senior.

I am personally in favor of the strong product management role for the Product Marketer. While in the early days of a company, the CEO or VP Marketing in a startup may have to wear multiple hats; eventually, you need someone to “own” the product.

There are, however, some clear rules that can help make sure that product management works well.

First, you need to plan. If you don't know what you are building, then it is hard to staff for the task, monitor the task, or have marketing work in parallel.

Second, you need to document. While I am not in favor of the traditional waterfall approach to software development, documentation is critical. My favorite approach to documentation uses two tricks. First, a company should write the manual before it writes the code. Second, programmers should document what they are going to implement before they program.

Just as importantly, the product development team needs to develop use-cases. These are the situations and usage patterns that the software needs to be designed to support.

In an ideal world, a specification should make the job of the developer easier. It should make his job one of implementation rather than iterative design-questioning-validation-redesign. However, in the real world, there is always input from developers. The secret is to harness the insights the developers have without losing control of the project.

Constraint Management

Inevitably, there is pressure to go to market. Some joke that you can have two of three software project characteristics – high quality, low cost, or rapidly developed – but you can never have all three.

The only way to come close to having all three is *reduce the scope* of the project. Software projects are always higher quality if their scope is restrained. I refer to this as Pareto's Law of Software. Twenty percent of the functionality will account for 80% of the usage and benefit. So start with the 20%.

Productizing While Bootstrapping

Software companies often finance themselves through a series of customer contracts. This bootstrapping approach is appealing because it reduces the amount of equity that a company needs to raise and avoids dilution of the equity positions of the founders and initial investors.

The downside of bootstrapping a company is that (1) the culture of the company begins to change into a more short term oriented, consulting culture, and (2) finishing the

Turn Around!

product becomes difficult, because of continued new requests from new paying customers.

Generally, the only way of productizing a set of tools being used in consulting projects is to set up a separate team. But even with this approach, it can be difficult reconciling diverging code bases without a very strong architect, capable of staying on top of multiple projects.

Productizing With Venture Capital

With all the problems of productizing when bootstrapping, one might think that the obvious solution is to raise equity. However, a brand new set of dangers emerges when venture capital is available.

The advantage of bootstrapping is that the customers keep you honest. Good customers actually help you develop your product's capabilities and can really focus on the Pareto functionality.

The disadvantage of not bootstrapping is that companies' managements can become arrogant and spend too long building a product, before seeking customer validation. This is almost always very expensive mistake.

The challenge with venture backed software development remains the same – develop the minimal functionality and get it used quickly. Even if you have the resources to build more, the benefit of usage is always high.

Do Usability Testing

There is only one rule here. Do usability testing. You will be amazed at what is not obvious to users.

Launching Products Takes Money – More Than It Took to Develop the Product

In the 1980s and early 1990s, the rule of thumb for packaged software was that it takes about \$5-10 of marketing and sales for every dollar of R&D. The Internet has given some the illusion that this kind of ratio has disappeared, but in my view it is still true.

Developing a piece of software is a bit like having a baby. Giving birth is only the beginning.

Clearly, the more that a company bootstraps its growth through consulting contracts, the less marketing it will have to do. But, generally, there is significant investment in (1) the sales and RFP (Request for Proposal) process, and (2) the need to customize the solution to win the project.

Obtaining leverage from the software when you are constantly adding features that lack broad appeal has a hidden set of costs.

Turn Around!

Software Margins Aren't as High As They Appear

Once software is developed its margin is normally considered to be high. Typical figures are in the 85% plus range. But in reality, this is only the incremental margin.

However, in my view this is a misleading number that erroneously underestimates the trailing cost consequences of any closed sale.

The more important margin numbers have to take into account three additional sets of costs:

1. The sales volume and resulting sales costs necessary to amortize the R&D.
2. The cost of selling the software.
3. The cost of supporting the software.

Why the Sales Cycle is So Important

Most software companies go out of business because either (1) their sales cost is too high, or (2) they never achieve enough volume to recoup the cost of developing the software.

A simple example will illustrate:

- Assume that it costs \$1M to develop the software.
- Assume that the assumption is that the company can sell 1M copies.
- Then the amortization of the development costs is \$1 per copy.
- If however, the market is too small, say for example that the volume sold ends up being only 10,000 copies, then the amortization of R&D is now \$100 per copy.
- If the product has been priced on the wrong volume assumption, say at \$50 per copy, then the company will never recoup its investment.

Because software products tend to have short life cycles, the other constraint is time. You only have so much time before a software product becomes “stale dated”. The Window for software sales is normally small, typically less than two years.

If a company fails to recover its costs on its first generation, it is unlikely that a second generation of the product will help the company recover unless dramatic changes occur in its market or strategy.

Sales Cost

Even if the company does get the volume right, a sales cost that is too high can also kill a company. While it is difficult to forecast a sales cost prior to product launch, it is an important exercise to have worked through the sensitivity of your cash flow to:

- Number of sales representatives hired.
- Number of prospects approached.
- Number of leads qualified.
- Conversion-to-closed-sales percentage.

Turn Around!

- Average dollar size of sale.
- Cost of delivery per sales.

Variation in the length of cycle is as important as the yield from the sales process.

Sales Forecasting

Sales forecasting is at the heart of survival for any software firm. There are essentially three useful ways of thinking about sales forecasting – yield, bid conversion and installed based modeling.

- For the low priced product, you can take a statistical **yield** approach to sales forecasting. This means you can use traditional direct mail models for measuring the cost of contacting a prospect and a predicted yield. The techniques for doing so are well established and can be applied to direct mail, web sites, e-mail and direct response advertising on web, TV, radio or whatever media you are using.
- For higher end software and early stage companies, as with most industrial products, the task is a bit more complicated. The first problem is that B2B oriented software often ends up being sold to a wide variety of customers, who buy for different reasons, at different stages in usage patterns and with different installed bases of hardware and software. There is little in the way of sales pattern.
- For many software companies, the calculations need to be based on probabilistic modeling of the sales funnel (the *probability* times the *dollar value per period*). A certain amount of effort and dollars will generate so many qualified prospects. Contact with qualified prospects will generate so many proposals. Proposals will have a success rate and average revenue size, with a trailing revenue and set of costs over multiple periods. There will be a certain rate of attrition between those that buy the software and end up using it successfully. Of those that use it successfully, there will be a certain percentage who go on to create repeat sales.
- For existing companies, the relative importance of maintenance and consulting revenues needs to be modeled along with the new sales. In the more mature business, installed based sales models are also likely to be useful. In other words, you need to project your penetration by type of platform and software usage of customers using the class of software you sell. The marketing challenge here is brand switching.

And of course, sub-models are required to model different strategic alliances, distribution and OEM distribution deals.

Revenue modeling is further complicated by the need to maintain multiple pricing structures. While legislation in the US does not permit discriminatory pricing for customers buying similar amounts of products or services, there is often wide latitude in the pricing used by software companies.

Turn Around!

For example, in one ASP company, it became very clear to us that while we had a basic transaction cost charge that varied based upon volume, it was necessary to offer two different types of pricing schemes:

For the very large company, we need to fix their budget and offer a flat price that would cover all of their anticipated volume if we wanted to get the business.

For very small transactions, we had to vary our price to reflect the fact that a smaller transaction could not justify the transaction fee of a larger transaction. So our initial transaction model needed to be adopted in the light of customer economics and negotiating power.

Cash Flow Forecasting

Along with revenue forecasting goes cash flow forecasting. For a software company that is up and running, the projection of sales that will actually close in the short term is a key activity.

The rule here is (1) to be on top of your cash flow at all times, otherwise money will go out the door faster than it comes in, and (2) you must have a forward looking projection that is continually updated.

Because sales people are by nature optimistic, it is extremely critical to have harsh criteria for accepting a revenue into your cash flow. One useful technique is to base your spending upon signed deals and separately do probabilistically weighted schedule of possible deals over the forecast time period.

Support Costs Matter

Many high tech businesses get into significant trouble because of their support costs. The approach I like to use, is to model support costs before you launch the product in a big way. Tools now exist that allow the modeling of sign up and support costs. For a tool in this category, visit www.primarymatters.com .

Or if the product is already completed and launched, then it is important to track support costs and use information about frequency and time to solve as a major input to future product development priorities.

Timing Is Critical

There is a lot of software in the world. One service that I use (www.softsearch.com) tracks over 130,000 pieces of software. So, understanding your market and the timing of your launch is critical.

Fundamentally, competing with established brands is difficult. It can be done, but because large software categories such as integrated productivity software (e.g. MS-Office, Sun StarOffice, Lotus SmartSuite) have high barriers to entry, new entrants such as Sun have to change the rules of the game in order to succeed.

Turn Around!

Launching new product categories is difficult for other reasons. Often software developers will coin a new “buzz word” to claim the attention of prospects. Sometimes the terms are adopted; sometimes not.

New categories of software generally require market education. And market education is expensive. So sometimes, the way to launch a new product is to sell or give away something narrower, where the decision to adopt is not seen as threatening or requiring a decision. Once the initial wedge has been established, additional functionality can be sold. This sales approach often requires backward compatibility to the previous generation of products.

Balance Installed Base Marketing vs. Customer Driven Requests

For any tool vendor, a major challenge is often the competing demands of installed base marketing vs. consulting. Consulting clients may or may not be typical of the rest of the market. So a product plan needs to be created to counterbalance the requests of paying clients.

In more mature technology markets, a critical task is to make sure that the software functionality developed will address the largest possible installed base of users whom you are attempting to switch. Examples of such decisions include the following types of functionality:

1. Data importing.
2. File conversion.
3. Operating system compatibility.
4. Hardware compatibility.
5. Support for particular application packages.
6. Support for standards such COM, CORBA, Enterprise Java Beans, XML, SQL.
7. Support for MS-Office standards such .csv files, HTML, .rtf, .xls, etc.

Talk and Listen Closely to Customers

One of the illusions of many first time entrepreneurs is that most ideas come from the founders of companies. In fact, most innovations come from customers.

Spending time living with customers is the single most important source of product innovation once the initial framework for the product has been developed.

Good customers are also practical. They understand their users and often are surprisingly willing to put with clunky features because their eye is focused upon achieving a quick small win. They are prepared to evolve the project later.

Working Capital and Relationship Profitability

Companies with no financial constraints are few and far between. There is always a dynamic tension between two sets of goals.

Turn Around!

Get Customers

The first imperative is to get customers. Because software is typically a multi-year relationship business, some companies are tempted to seduce their customers into a relationship on the assumption that they will make it up in the subsequent years of the relationship.

Build a Business With Negative Working Capital Requirements

The most successful high tech companies in the world are in the enviable position of having negative working capital requirements. They pay their suppliers more slowly than they collect from their customers. Apple, in its early days, Microsoft and Dell today, all have this characteristic.

There is no hard and fast rule on this dynamic. My prejudice leans towards having a product with such value that you don't have to discount the product in the first year to buy the relationship. But if the numbers work the other way, then it make sense to reduce sales costs and achieve quick market share with an easy entry price.

You Need Reference Sites

Reference sites are key to any software company. However, I am generally against giving free software to reference sites. People who request free software rarely commit to using it and put resources behind it. By all means, over-service a reference site. You can even do deals where they get a limited royalty to recover their investment in your software, but for them to use your software, they need a budgeted project. My view is: make them pay and give them great value.

Pricing Should Be Based Upon Economic Value to the Customer

Some strategists don't like software because they feel that software always trends down to a zero price. I disagree. Software should always be priced based upon the value created for customers (and their clients).

Value can be generated in many ways. Some of the obvious ways include:

1. Reducing the cost of an activity.
2. Accelerating time to market for the customer to preempt competition.
3. Creating new revenues for customers.
4. Reducing client loss or increasing repeat purchase for customers.
5. Reducing the cost of an activity and increasing its quality.
6. Improving the timeliness of decision making.
7. Creating new sources of customer value that allow the user of the software to make more money off his clients.
8. Improving the resource allocation decisions of customers so that they can change their mix of activities.
9. Solving regulatory or liability requirements.

Turn Around!

10. Providing a solution in a software category that the customer cannot find staff for.

Setting prices for products should, therefore, always be based upon knowledge about the customers' economics, constraints and opportunities.

Pricing also has to take into account the multi-period nature of the relationship with a customer.

Pricing vs. the Competition

Because repeat purchase and multi-year relationships with customers are the basis for making money in most software business, competitive pricing issues will affect the pricing based upon economic value to customers.

The fundamental premises of competitive pricing are the following:

1. *Fast to market with high quality product* often leads to higher market share, higher profitability and return on investment. Fast to market with poor product leads to branding your product as a dud.
2. *There are economic advantages to being a market share leader* or at least a strong number two. This advantage should eventually translate into a higher perceived value and higher pricing.
3. *Classic experience curve pricing* prices based upon where your price is going to be if you achieve a certain market share not on where your cost structure is this year. Using this experience curve pricing requires that you be well funded.
4. *Prices in most software markets tends to drop over time.* Business plans based upon rising revenues per customer need to be justified by dramatic increases in functionality, additions to product families or revenues from services, support and training. So premium pricing is always relative to the drop in pricing.
5. License revenues tend to be the measure of software sales success. If a company is achieving sales growth mainly due to maintenance revenues, it is not a healthy company.
6. If you achieve *category dominance*, the rate of price drop will be less and can even reverse.

Specialist Software and Tool Firms Have a Good Future

Fundamentally, *software is a scale game*. Software is complicated and requires both expertise and descent of a learning curve. If you can amortize your R&D costs over more customers than a single customer developing for its own organization, you will always have an economic advantage due to scale.

If anything, this trend is increasing. There was a period in the 1980s when large multinationals like Exxon actually developed tools that today would be categorized as spreadsheets. No non-software firm in its right mind today would develop a spreadsheet. First, the product is late in its life-cycle. Second, increased competition tends to force specialization on both customers and suppliers. And the more advanced an economy, the higher the degree and web of outsourcing. As more and more software areas emerge, it is

Turn Around!

increasingly difficult for customer organizations to do more than implement or integrate. In many cases, they will prefer to out-task and outsource even the integration, because they lack the ability to maintain software and staff.

As a result, many tool vendors will increasingly be forced to develop professional services and outsourcing capabilities.

The only caveat is that the tool or solution must be customizable sufficiently easily that the cost of the relationship with the tool/application supplier still creates more benefit than a self-developed tool. For most software categories today, the scale relationship holds true.

Time-Based Competition

While scale is critical, so is the rate of learning and product revision. In the automobile industry, the Japanese penetration of the US market has been based both upon high quality *and* shorter product life cycles. Faster life cycles allow marketing organizations to learn faster what users want and keep upping their value. Size helps organizations afford this frequent revision.

But for frequent revisions to work, there needs to be a corporate culture of speed.

One of the significant impacts of the Internet is the ability to monitor usage patterns of users on a real time basis. Provided that privacy concerns are dealt with (and this is no small task), information about support problems and usage patterns, can provide the basis for significant competitive advantage.

Do Things in the Right Order

Most startups get into trouble by doing things in the wrong order. Whether it be product management, product launch, financing, sales force development, public relations, a good startup has to plan. Sequencing makes a difference.

The plan may change, but planning ensures that work can be orderly and that when the plan must change, the company already understands the dependencies and staffing consequences of changing the plan.

Time to market does matter, after all.

Business Models

Software companies need to model themselves. Investors typically request a bottom-up financial model that demonstrates the revenues sources for the company.

It is equally important to model the cost side of the business. In my experience, there are four key modeling exercises that companies need to do:

Turn Around!

1. Model the customers' usage patterns so that the value of the software to the customer can be estimated.
2. Develop a pricing model that will produce the most rapid adoption of the software.
3. Model the costs of signing up and supporting the customer over a multi-period relationship.
4. Model the cost of maintaining and rewriting the existing functionality of the product and project by feature, the cost of developing the new functionality projected in the product plan.

Turn Around!

Chapter 2: Building a Compelling Software Capability

The research on new product development is very clear. The single most important factor in launching a product successfully is offering a high value product. Such a product must be differentiated and the value perceivable to the buyer.

The question is: “What is a high value product?”

There is of course, no magic answer to this question because value changes as a market evolves.

I paid a premium for my Palm Vx PDA (personal digital assistant) because it fits in my shirt pocket. For me, small size was one of the aspects of value for this product category.

In fact, the Palm is an interesting product because unlike every other computer I have ever bought, the speed of the processor (a traditional measure of value) was irrelevant for me. In fact, so was the ability to add software to it. My other reason for delight in the Palm was the ability to avoid carrying my laptop with me and having to wait for it to boot. I eventually discovered another value of the Palm was that it became a back up for my contact list. It was also easier to enter addresses on than my cell phone.

So fundamentally, a software product must **solve an important problem** for a customer in a powerful way. Incremental functionality is a tough basis to grow a business on in a world of almost unlimited software availability. The Palm Vx solution was lighter, turned on faster, is simpler to use, and easier to enter data on. It also fit in a shirt pocket and looked sexy with a chrome case.

What was important about the Palm purchase was that it went to the core activities of my profession – keeping track of people, contact information, events and emails. For my retired neighbor, Richard to whom I gave a diary I had accidentally bought, a Palm Pilot has little use.

So, another aspect of value is that is always specific to a segment in the market.

Another aspect of value may sound a little fuzzy, but it is the notion of **synergy**. For many pieces of software, a solution that solves a number of different problems at the same time produces multiple values.

In one company that I headed (www.alacrity.com), we substituted a very powerful object database for relational database tools and were able to use it for storing objects without decomposing them (saving us development time and testing, increasing reliability); we used it as an application server, so we could build *light* (server-centric) applications; we used it to create a distributed set of servers for an information warehouse; we added OLAP functionality, using a superior model to those currently deployed: and we build other applications around this core technology.

Turn Around!

In contrast, traditional solutions would have required support three or four different data architectures. By the time we were finished, our developers would have had to be tortured by the medieval Spanish Inquisition to give up on the marvelous development environment (www.gemstone.com). Elegance, synergy, ease of development, all contributed to the perception of value.

There is another aspect to creating a compelling software product and that is the first experience of use of the product. Using a mouse with a graphic interface was a novel concept in the 80s was a new concept for computer users. Cleverly, most vendors included games like Reversi or Minefield, or painting programs like McPaint, which allowed immediate use of the mouse, immediate visual feedback, and a sense of accomplishment and mastery. Such “mastery” really only involve learning how to click on an icon, to draw by holding down the mouse button and how to use pull down menus. Not much to learn, one might say, but it was and is an effective learning tool.

Learning theory suggests that rapid early feedback leads to faster learning. Good software provides feedback so that the correct and incorrect behavior of the user is quickly learned.

So it is my belief that value starts with **ease of use**.

But ease of use is not enough. **Ease of installation** is a non-negotiable aspect of value. And installation clearly needs to be done before use.

But software users change as they learn more about the software. So, a second element of value I believe is providing users with a **mental map or metaphor** for thinking about their work on or interactions with the computer.

Often, computer software has so many tools that actually understanding the connections between the tools or the menu options that influence a task is a mind bogglingly difficult task. Take for example the setting up of networking on a Windows machine. I don't know how many hours I have wasted on this task. There is no one place for managing all the information and no unifying metaphor.

Another aspect of software is that software is increasingly used in conjunction with other software. The ability to have software work together with other software is a key requirement. If you are selecting software development environments, it is not enough to have the standard class libraries, documentation tools and programming language. Increasingly, you need the hooks into standards such COM, SQL, XML, HTML, EJB, etc.

This **integration requirement** is often forgotten by startups. I worked with one founder of a company. He had build a transaction oriented e-commerce system for managing supplier purchases but had not addressed the set-up issues. There was no easy way for the new user to set up the data in the system.

Turn Around!

In the quality literature, quality is often talked about in terms of meeting customer expectations or **delighting customers**. To me, some of the most exciting software products are those that delighted me by giving me functionality that I had not expected. When I bought my first digital camera, I was delighted by the ability to use various software packages to edit and transform my photos. I was even more delighted to discover that apparently bad shots could be fixed with various transformation functions. I was even more delighted to discover the various artistic transformations, such as creating an Impressionist version of my photo.

Time saving is frequently a major feature of software. E-mail, digital photography, editing documents, faxing software, spreadsheets – they all share a major time benefit.

Cost reduction is a major benefit too. The tasks that I do on my notebook today would have required an office staff twenty years ago. Even the simple act of creating a slide presentation used to be an expensive process when slide shows had to be done and photo slides and used in a projector.

Process vs. Output Benefits

Many of the benefits of software fairly obvious – faster, cheaper higher, more affordable, high quality outputs.

But over the past two decades, I have noticed that an increasing number of software packages have explicitly or implicitly another class of benefits. These benefits are procedural in nature. Procedural benefits may be harder to explain, but they can have *order of magnitude* benefits to clients.

If you think about a spreadsheet for example, there are numerous ways of setting it up. Bad spreadsheet users conceal all the relationships in hard-to-find and hard to read formulas. Good spreadsheet users put the variables in cells where they are visible (and so are easy to validate) and more easily changeable. The spreadsheet is in effect a blank canvas with little in the way of implied methodology for best practices in building a spreadsheet. (Strictly speaking, a spreadsheet is a deterministic model that tends to guide users away from stochastic or Monte Carlo approaches, but within the deterministic modeling framework, it provides little in the way of methodological guideline.)

Spreadsheets are immensely powerful *ad hoc* tools, but require genuine programming talent to turn into something usable by multiple users.

In contrast, many other types of software have an implicit methodology that the user must follow in order to solve his problem. Project management is one example. Project management software is typically very difficult to use. It requires understanding of the process of project management and is, I think most people, would agree a much less satisfying class of software than a spreadsheet. Part of this lack of satisfaction is the

Turn Around!

assumption behind most project management packages that the user knows how to do project management.

Software that guides a user through an interaction turns out to provide some enormous benefits. At a former company, Alacrity Inc., we discovered that our strategic expert system was valued more for the disciplined questioning process than it was for the answers. What we eventually figured out was that the process of being debriefed by software in a Socratic or structured question and answer dialogue was just about as effective as a human asking the group of users questions.

We had originally focused a great deal of attention on the answers provided by the expert system, but what we discovered was that we were actually selling a methodology – a way of profiling and analyzing your business.

I have since observed in many products and projects that the same is true. As software and the issues software packages address become more complex, the task of providing a set of functionality to solve the problem is increasingly inadequate. Software developers need to think in terms of providing methodologies for supporting the set-up, usage, change management or configuration process, depending upon the nature of the software.

Turn Around!

Assessing a Compelling Software/Service Product

(Score up to five points for each category where 5 is best and 0 is worst ranking.)

1. Benefit message easily understood.	
2. Users know they have a problem.	
3. Benefit is so high relative to cost, that the purchase process is an easy decision.	
4. The software is easy to find.	
5. The software is easy to purchase.	
6. Installation is nonexistent, or flawless and easy.	
7. Good friendly knowledgeable support is available.	
8. Support queue times are short.	
9. Migration and data exchange issues are handled well.	
10. Synchronization issues are nonexistent or handled well.	
11. The risk of software or service usage is perceived as low by users.	
12. Usage of the project does not reduce user capabilities.	
13. Usage of the project does not complicate users' lives significantly.	
14. Users can talk about the benefits of the product to other users easily.	
15. Users encourage others to adopt.	
16. Users want to upgrade at a high rate.	
17. Users are emotionally involved and committed to usage.	
18. Opinion leaders appreciate the product.	
19. The product/service is perceived in a leadership position.	
20. Conventional wisdom about the product category is positive.	
Total out of 100 points	

Product Assessment Rankings

Range	Assessment	Comment
0-50	Unattractive	Difficult to make successful
50-74	Possible	Fix shortcomings
75-89	Good	Execute well
90-100	Excellent	Good chance of success

Turn Around!

Chapter 3: Common Problems in Software Teams and How to Resolve Them

One of the most common problems in software projects is lack of experience. Many people are accustomed to using or even building software. Some have become adept at using tools like spreadsheets and even programming. They become enthusiastic about the business potential for software, and before you know it, there is a software business started.

Most software projects fail. Most software businesses fail.

If you start with this assumption, you will begin to understand why having a professional project manager is often very helpful. If you are new to software development, I recommend very strongly reading a book I co-authored on advice for succeeding with software projects called *Riding the Tiger* (Davidson, Gellman and Chung, Jist Publications 1999 in the US and HarperCollins Canada 1997 in Canada).

The Interface Problem

In my experience of start up software companies headed by non-technical CEOs, the typical problems come from the interface between the vision of the founder, the lack of a methodology for prototyping and specifying, and frequently inappropriate tool choice.

The typical range of wasted budgets that I am seeing in companies, when I first start to work with them, ranges from 90% down to 50%.

Now to be fair, in most software projects, there is frequently a learning curve, particularly if the problem is new. Most people experience around a 25% wastage on their efforts. And for novel software, a certain amount is reasonable. But over time, companies should be attempting to advance up the software capability maturities model (see the web site of the Software Engineering Institute at Carnegie Mellon for more details at <http://www.sei.cmu.edu/>.)

We claim in our book, *Riding the Tiger*, that it takes three times to get a piece of software right. So even if your first generation projection works and 75% of your budget was well spent, you still have to expect ongoing revision. Generally, Generation 3 of the software requires a fairly substantial rewrite and often, major architectural change.

An Example of a Failed Development Process

So, the amount of money spent on wasted software development is very high. One company whose product development process I had to turn around, had spent three years and four million dollars on product development and had nothing of value to show at the time that I was hired.

Turn Around!

Reasons for their failure started at the top: senior management had no ability to lead a technical project. Their early CTOs were not of high quality. And to add misery to madness, the company hired low cost Eastern European programmers with poor language and communications skills. They made poor choices of technical tools and did not train their staff. Add the lack of a clear objective as to the functionality of the project as an additional negative. There were at least two radically different objectives during the project. It was no surprise that the company had been unsuccessful.

One of the difficulties in managing companies with failing projects is that poor specification and poor design generally results in untestable software modules. The excuse is always that a module can't be tested because something else isn't working right.

In these situations, you are generally better to restart the project. Develop a specification that can be built in two-week chunks with reviews every two weeks of working components. If your architecture is not modularized into a component framework, there is probably little chance of success without replacing your architect.

Long Development Cycles

Long development cycles for deliverables are a sign of failure. Period.

It may take a long time to build an entire project. In large projects every project must be constructed so that there are regular and short deliverables. If a developer says it is going to take 2 months to build something, my advice is don't believe it. Shrink the deliverable and review it early. Have it tested by someone who did not build it.

Relying Upon the CTO/Architect to Specify the Product

Specifying software is very hard. Frankly, it is a task that practically no one likes to do. Often the CTO or Architect will end being the specifier of the product. He in effect, is doing it out of desperation. Nobody else will step up to the wicket.

Unfortunately, it is the rare CTO who is also a domain expert and user of the software under development.

Even when the CTO or Architect is competent to specify the product, the consequence is that the product gets built without documentation. So documentation needs to be added later, and often the process of documentation reveals usability issues.

Generally the architect-driven software development project is the most successful in producing a working piece of software, but it is the rare project where the architect has enough knowledge to be the specifier of the entire project. Typically a *visionary domain expert* is required. Only with more technical projects can the architect also be the domain expert.

And increasingly projects require integration of functionality that would stretch the ability of the most talented architect. Software is increasingly integrated and the act of

Turn Around!

software development increasingly involves incorporating other people's standards, software and ASP services. It doesn't quite take a village, but it certainly takes the magnificent seven.

So practically all software companies need to devote more time to technical design and technical product management that they feel they have resources for. But sharp and crisp early definition of a product can speed time to market, lower development costs and dramatically increase both the probability of commercial success and market share obtained.

Documenting in Parallel

In turnaround situations, I have had to develop and document in parallel, but frankly, it has never been a very satisfactory solution. In most cases, I tend to think that a firm is better stopping development for a period of 1-2 months and doing a crash project to document the Pareto functionality and architectural framework.

In that period of downtime, programmers can either document what has been done already or receive training and the project will probably end up being completed faster. Documenting after the fact generally demonstrates how poor the software code is unless the developers were unusually talented.

A Key Assumption About Software Development

Perhaps I should reveal a key assumption that I hold to be true about software development, one that is not understood by most managers: my experience and those of others knowledgeable about software shows one thing very clearly.

Some developers are a hundred times more productive than other developers.

They have a clarity and quality to their code that puts them into a league that is so different from ordinary developers that keeping them productive is critical.

These superstar programmers are not hackers. They don't create solutions that barely work, and which are hard to maintain. They are, rather, visionaries that create frameworks, architectures, modular code that magically seems to have a long life and the ability to provide a robust and scalable frame to which others can add.

Good software development generally only ever happens around these superstar developers.

Unfortunately, many developers think they are superstar developers, but never end up delivering. Delivery is, after all, what being a superstar developer is all about. The good superstars want to see frequent deliverables. They too have learned that there is no substitute for working code.

Turn Around!

Tying the Business Model to the Technical Development Strategy

Basically, just about anything can be programmed these days. But software can be optimized for different aspects. You can optimize your software for rapid development, for flexibility, for speed, for maintainability, just to name a few issues.

It is, as a result, really important to think about a product specification from a multi-period perspective.

One telephone company that I worked with sold primarily to large corporate customers. They had, in prior periods, hired a systems integration firm to build them a billing system. The system was able to handle a huge amount of flexibility in billing approaches to match the almost custom relationships that the company had with its customers.

When the company won the right to enter the retail market, they discovered that processing each simple retail customer account took many minutes. They had to purchase a completely new system. The old one had not been optimized to take into account speed and volume of processing.

The Back of the Envelope Calculation

Way back when I was in business school, one of my manufacturing professors used to talk about the “back of the envelope” representation of the problem. What he meant was the ability to reduce complexity to a simple representation so you could understand the problem.

Practically every software project should have a back of the envelope projection done on it. In most software projects, scalability ends up being an issue, so it is really important to figure out up front, where you are likely to run into bottlenecks.

Because software is perceived as esoteric and difficult, many managers will refrain from tackling this exercise, but it is essential. If you don't have the skills personally, you need to bring in an expert. A half-day of his time may be the single best investment that you ever make.

A simple example will illustrate. Let's assume you are building an OLAP (on line analytical processing or multi-dimensional spreadsheet) application. OLAP tools allow you to create an n-dimension view of a problem. So, you might consider building a model of a business where one dimension is the budget income statement, a second dimension in the budget vs. actual dimension (perhaps with variance, YTD budget, YTD actual, YTD variance), the third dimension might be years, the fourth dimension might be countries in which the firm operates.

The size of the multi-dimensional model is calculable:

$$\begin{array}{c} \text{Number of lines in budget (Say 50)} \\ \times \end{array}$$

Turn Around!

Number of budget, variance columns (Say 6)

X

Number of years displayed (Say 5)

X

Number of countries (Say 10)

The OLAP model is 50 X 6 X 5 X 10 in size or 15,000 cells of data.

If you decide to expand the model by adding 20 products and 5 segments in two separate dimensions, the 15,000 cells suddenly grown to 15,000 X 20 X 5 = 1.5 million cells.

Simple exercises like this clearly affect the design of a system. In this case, a reduction in the budget detail would have significant pay-off. Another simplification is compress dimensions and combine the data from several dimensions into one. The benefit is that exponential growth is limited.

There are numerous examples in software design where this kind of analysis is not done until late in a project when speed degradation is an issue. A good architect will anticipate such problems, because he will have run into them before. But a manager must always check that the issue of scalability has been thought about early in the project. It is always much cheaper to do it early in the process.

Turn Around!

Chapter 4: Marketing and Sales

Hiring Good Marketing and Sales People

Some technical founders of companies think that hiring good marketing and sales people is a mysterious task. In my view, it should not be.

The characteristics of good marketers is that they understand how to translate a feature set into something that end-user customers can get excited over, excited enough to buy. Everything else is execution. And while execution is important, if the marketer can't pass the first hurdle, no amount of execution is going to help. Track record is no predictor of success. A successful product marketer from outside the software business may never be able to make the transition because he is overwhelmed by the technology.

Good sales people are, above all organized. They are disciplined, keep track religiously of their prospects and can at all times tell you about their sales funnel – the ranking of prospects as they move from target to qualified, from qualified to interested, from interested to intending to buy, from approved to purchased, to repeat purchase status.

Good sales and marketing people both know how to prune effectively. The marketer will force the company to focus its efforts. The sales person will focus on those who are going to buy most quickly.

Many technically-oriented founders are upset by this pruning. They interpret the pruning as walking away from an opportunity, but marketing and sales are both about prioritization. You can always target other segments later.

Managing the Vision; Listening to Customers

One of the most important roles for the CEO or in a multi-product company the Product Marketer is managing the trade-off between the vision and the market feedback.

Without the vision of what the software solution is growing towards, software companies tend to become me-too software producers. But with too much reliance upon customers who cannot visualize the vision, a company can lose sight of what in the long term is going to make it great.

There is no simple answer to managing this dilemma because a company's ability to create its vision is a function of its financial resources. And if it is not selling what customer want, it will in most cases not be able to fund the necessary R&D to create the over-aching vision.

Pioneers vs. Settlers

In hiring marketers and sales people, one important rule to bear in mind is that there are pioneers and settlers. The pioneers are the people who like to develop and create new

Turn Around!

software markets. They like the fact that the territory is uncharted. They don't mind, in fact, they relish opening up a new industry.

In contrast, the settlers are happier doing what they have done before, selling to the same industry. Lack of fit in this area can lead to major costs due to turnover; and turnover is always expensive.

The Software Sales Presentation

Over the past two decades, no subject has been more controversial than how to present software in a sales situation. Everyone seems to have a different view on how to present software to prospects.

As someone who has inflicted more than his fair share of Microsoft PowerPoint and software demos upon customers in multiple languages and countries, I feel very strongly that software demos are a very poor way of selling software.

I, personally, have bored more people with software demos than I care to think about. Now, I would like to offer as an excuse that in the early days of software development, the LCD panels (the ones that sat on top of an overhead projector) were not exactly vibrant. My first In-Focus panel had two colors – blue and green. And in those days, we were still using character based DOS.

But, I believe that customers typically don't buy based upon software demos. Oh, yes there are the odd pieces of functionality that are very exciting. The first time I used a mouse to do a drawing program, yes, I was inspired. The first graphic based spreadsheet I used was exciting. The demonstration of how to use a WISYWG word processor where what you saw was what you got – it was impressive. And I still find voice recognition pretty amazing.

But earth-shaking, paradigm-shifting software is not the norm. Most people are selling applications with less novelty. And people are no longer wowed by what they once were.

My rule of thumb is that you should be able to sell software without showing it. If you do have to show it, you should show as little of it as possible.

The fundamental reason for this restraint is that software demos tend to focus on showing all the marvelous features that have been designed into the software. And most first time viewers of software can barely figure out what is going on and so they become confused and switch their minds off early in the presentation.

It is the rare software purchaser that makes his decision upon seeing all the features in a software product. Rather, there tend to be one or two capabilities that do excite them and those are the ones that the presentation should focus upon.

In fact, what makes software attractive to customers other than the one or two sexy features is typically the architecture or workflow of the product. Not because architecture

Turn Around!

or workflow sell, but rather they can support the **benefit** claims that really drive the emotional decision to purchase. So often the best way of selling a complex software product is to show its architecture and link the information to the benefits.

My test of a good sales person is “Can they sell without showing the software product?” If they can, they understand how to sell because in selling a piece of software, the ideal sales process is always:

1. Understand the problems of the prospect.
2. Indicate that you can solve this problem and express the benefit.
3. Prove that you can do it by referring to other users, white papers, cost-benefit studies or in the worst case, show the software.

When I suggest showing the software, there are in fact three basic approaches to “showing software”

The first is to have the live software on your machine with an example that the sales person is thoroughly trained on.

The second is to have a crippled version of the software with an automatic running capability.

The third is to use a PowerPoint presentation or similar animation package to show the screens. This approach is inexpensive and very reliable. It requires much less training than the having a full version of the software.

Because developing demos is so time consuming and costly, I am generally in favor of developing low cost slide show based presentations until the organizations knows what it takes to sell. The effort that would otherwise have been going into product development always seriously detracts from the development process and generally has a huge cost in delaying time to market.

The Lemming Industry

Selling software in some industries is a painful process. If your software requires organizational change, then one of my least favorite industries is the financial services industry. Financial services organizations tend to be large; they tend to be skeptical of any solution developed by a small software firm that requires organizational change. And of course, practically all software startups are small. So while they will often meet with, and chew up huge amounts of sales time, the chances of them actually buying are very low.

Except if you make a sale to one of their big competitors. Then with that endorsement, the lemming button switches on and now everybody needs your software. They will still chew up enormous amounts of sales time with RFPs and RFIs, but now they will potentially buy.

Turn Around!

The secret with selling to a Lemming Industry like financial services is not to. Sell to someone else first. Just maybe, they will be impressed by sales to large but non-financial services companies and you can waste less time.

But remember: high sales cost is the killer of small software companies.

Public Relations

The conventional wisdom on public relations (PR) for software companies is that there is an order in which you do your PR. You start with the influencers (columnists, consulting firms like Gartner Group, Meta Group, IDC, Forrester, Aberdeen, IDC etc.) and they do defining pieces. The defining pieces of analysis set the stage for the technical and business press. Eventually, in some product categories, stories migrate to the popular press.

All good in theory. Unfortunately, there do exist product categories, where this model was not followed. Examples include companies like Napster and Hotmail that are classified as viral marketing approaches. One user begets another user who begets another user and so on.

The problem for most firms is that they are offering me-too products. These products are hard to get press coverage for. Fundamentally, journalists need stories and most software has no story attached to it.

Software Evangelism

One of my favorite activities in a software company is evangelizing for a new piece of software or a new way of running a business. But it is really important to understand that the evangelist is not a sales person. The evangelist may influence clients through articles, seminars and key note addresses, but sales follow-up and marketing are still critical.

For the small software company, articles, seminars and presentations are exceptionally useful ways of getting the attention of prospects. However, it takes talent and commitment to produce useful material.

To be frank, most CEOs don't have the ability to be good evangelists. As a general rule, speeches by CEOs, particular CEOs of large companies are plodding and dull. They rarely can afford to be controversial. At best they get to make new product announcements.

In my view, the evangelist's goal is to change the way that prospects see the world, so that they will prefer the software offering of the software company he represents.

Marketing Software

It is fashionable in the software industry these days to think about the revenue generation side of the business as having five pieces:

Turn Around!

1. Marketing
2. Sales
3. Business development
4. Account management
5. Relationship management

Marketing's task is to defined the needs in the market place, translate those needs into a product that will create value for customers, position the product relative to alternative ways of solving the customer problem, and develop the materials for communicating about the product to influencers, customers, and prospects.

Sales is responsible for getting the orders.

Business development is responsible for negotiating leveraged relationships that can help gain access to new channels of distribution or large buyers who might incorporate software into their own product or services solution.

Account and relationship management are accountable for making sure big customers, distributors and alliances are being well serviced and opportunities are being captured. Many software companies that attempt to build their businesses through strategic alliances and distribution partners do not anticipate the amount of work necessary to make such relationships work.

In many cases, there is no point in entering into a strategic alliance with a big company unless they agree to fund the support staff necessary to service the alliance.

Some Marketing Guidelines

A big problem for many software companies is the confusion between the potential or total market and the target or served market.

Many software solutions can in theory be sold to many different types of buyers. However, the cost of selling to different classes of buyers can vary widely. So can the benefit that a software solution provides. Different classes of buyers may vary widely in their need for support. Different buyer segments may place different value on different pieces of the software and their buying cycle will be affected by their needs, value perception and purchasing power.

So the common argument in software companies is the one about whether to specialize in one segment or try to service more than one segment.

In a sense, this is the wrong question to ask. Most companies launch their software product with little idea of who will actually buy their product fastest. And one customer buying can be very misleading. I once sold a product into an industry, developed a great and profitable relationship with this single company. But I was totally unable to reproduce the success with other participants in the industry. I had lucked into the only innovator in the industry. It was a very costly lesson to learn.

Turn Around!

The innovators who buy your product will not necessarily be the mass market on which you will grow your company.

The innovators will look for you and look for your special features. The mass market will need to be marketed to, and will require the development of a brand that will reassure them in their purchase decision. In more mature markets, brand leadership and a low cost structure tend to drive software success.

A good book in the topic of migrating from early stage adopters to the mass market is Geoffrey Moore's *Crossing the Chasm*.

Test Your Results

My general feeling is that in the very early stages of product launch, you want to cast your net widely to get a sense of who is buying and for what reasons.

This is a particularly valuable period for a new firm. It is at this point that you can figure out whether:

1. You can make money selling a software tool.
2. The tool requires training in order to have successful clients.
3. The real sale is consulting and the tool is just a way of getting consulting.
4. The real opportunity lies in selling an application.
5. The real opportunity lies in a particular segment of the market.

Knowing what the customer will pay for is a critical insight. And as previously argued, sometimes software is the item that gets purchased. Sometimes software is the invisible tool for solving the customer problem.

The launch period is also a good time to test your pricing. Knowing how your pricing affects sales cycles is a measurable task in some markets.

Trade Shows

As a general rule, trade shows are a waste of time. I know that a lot of people go to trade-shows, but for the typical small software firm, it has to be a very specialized trade-show with real potential buyers. I say "real potential buyers" because many trade shows are visited by "tire kickers", in other words, people who really do not have any potential of being converted into paying customers.

Think too about the cost of a trade show. By the time you have planned the trade show, designed a booth, developed art work, printed brochures, rented the booth, paid for travel, hotel and lodging expenses, you have typically spent in the range of \$10-20,000 for a small trade show. When combined with the opportunity cost of staffing with people, the cost can be in the \$50-100,000 range.

Turn Around!

If you are going to do a trade show, the first important step is to get the breakdown of last year's visitors. Another good piece of research is to speak to a vendor (obviously not a competitor) who exhibited and find out what kind of attendees visited their booth and whether they saw any sales out of it.

Literature Development

Most small firms have difficulty developing the necessary literature for promoting their product. But the package is normally fairly standard. I like to develop the following pieces:

1. Web site content.
2. A fact sheet for the product.
3. A brochure. Sometimes more than one brochure is needed if several different classes of customers with different needs and language are being targeted.
4. One or more white papers that analyze the benefits of using the software. These documents are often difficult to do, but exceptionally critical sales tools.
5. Third party endorsement of the software from a major technology consulting firm or comments from customers.
6. Slide-show demos.

If you are a small firm with limited budget, I recommend not printing brochures but rather publishing them electronically in Adobe Acrobat format. PDF format is cheap to produce and can be changed easily.

Turn Around!

Chapter 5: Promoting the Web Site

Clearly, the larger the marketing budget you have, the easier it is to use “pull” advertising to drive customers to your web site to find out about your product or service. However, for companies with limited budgets and a product with international but thin markets – in other words, there are few customers in each market for your product, but they are spread out across the world – then the challenge is making sure that your web site is findable.

The secret to making your web site findable is to understand how people look for material on the Internet.

Most will use search engines. Some will come in through directories and consulting firms.

Understanding how search engines classify your web site is critical, because if you don't turn up in the first two pages of a search engine, your chances of being visited can get very low.

Making sure that you show up is an ongoing process not a one time event, that should be part of Marketing's mandate.

First, you need to put in place tracking software or services to make sure that you know who is actually visiting your site. Some services can tell you how the visitor found you – what search engine they came in from, what they were looking for, or whether they typed your URL directly.

Secondly, you need to make sure that your web site is using the words that people are typing when they look for your software. If you are selling “financial management”, then make sure the phrase is being used on your web site. It is particularly important that it is used high upon your first page and in your meta tags. The frequency of usage can also influence some search engines.

Metatags are the tags that many search engines use to help them classify your web site. It is important to know which of the metatags are producing traffic. If some are not producing traffic, change them. You can also run experiments by having different metatags on different pages and observe which produces higher yields.

A good validation here is to look at the source code on your competitors' pages and see what metatags they are using. Then when you search on these terms, you can compare how you rank with them. Different companies have different abilities to promulgate a new term. I did some comparisons for a company and discovered that IBM had coined a new term, so when you searched on IBM's term, it appeared as if IBM was the only company in the field. Merely adding the IBM “buzz word” to the meta-tags had a major impact on the client's ranking with search engine.

Turn Around!

Third, you need to examine whether anyone is pointing at your site. This is increasingly important because some popular search engines like Google use citation ranking or the number of sites pointing at as a measure of your importance. And the more important the site that points at you, the better.

If you are a site that no one points at, then you have to spend a great deal of time figuring out who might like to point at your site. Good “pointers” might come from consulting firms, directories and magazines. Often, these sites will also require that you point at them, so it is a good idea to set up a resource area of your site where you can point at directories, consulting firms and magazines. It’s a case of mutual back scratching.

Fourth, you need to register your site with the search engines. This is no small task because it is best done by hand. While there are tools for mass registering with multiple search engines and directories, my experience suggests that manual updating is more reliable.

While clear leaders have emerged in the general field of search engines. Tools such as Google, AltaVista, Lycos, NorthernLight are definitely leaders. But don’t ignore the specialized industry sites that through word of mouth end up being very powerful. They are frequently specific to your industry or product.

Turn Around!

Chapter 6: Defining Your Competition

Most startups are under the illusion that their product is so unique that they don't have any competition. If this is true, you may not be as lucky as you think. Often if you so leading edge, you will have difficulty educating the market about your product category. Ironically, it is sometimes easier to be a player in a market with competition. The challenge is merely to have superior features and services or a better pricing model; both are simpler challenges in many cases than educating your market.

But the reality for most firms is that you are competing with a wide variety of firms, products and services. In the Internet arena, it is particularly difficult to figure out what firms do. The result is that companies end up being your competition that actually have little direct overlap with your functionality and services.

As a general rule, there are typically five different classes of solutions that compete with a software company:

1. The internally developed solution.
2. Packaged solutions.
3. The systems integrator "assembled" solution.
4. Out-tasking solutions such as ASP or Internet delivered solutions.
5. Out-sourcing.

Understanding the advantages and economics of each of the approaches is often difficult without having staff, who have developed or sold such solutions. However, the following table provides a framework for thinking about the different classes of competition. When positioning the benefits of a particular approach, e.g. a tool vs. a package, a software firm needs to think broadly about its economic advantages and disadvantages. And from a marketing perspective certain segments will be more amenable to some types of solutions than others.

A good tool for doing broad scans of your competition is Copernic, available at www.copernic.com. This tool, which in my family was discovered by my web surfing father is an astoundingly useful meta-search tool for doing broad searches.

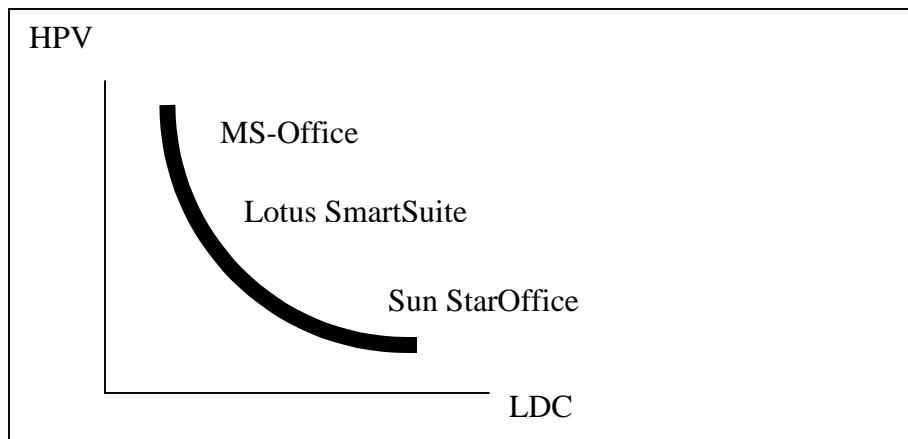
Turn Around!

The Economics of Different Solution Approaches

	In-house	Packaged	Assembled	Out-tasked	Out-sourced
Time to develop	Long, though productivity of object and component frameworks is changing equation	Short	Medium to long.	Short if the supplier is really knowledgeable in the area and has solutions already developed	Either existing code is taken over, or standard package can be used. So short.
Learning curve	Typically at least 50% of efforts wasted	Assuming a good package, very low.	Theoretically less, but many consulting firms have you pay for their learning curve	Supplier needs to learn customer business	Little
Staffing issues	Typically staffing is a problem	If good training available, few staffing issues. Turnover a possible issue.	Large risk that consulting firms use inappropriate staff	Generally supplier has good staff. Capacity may be an issue	Few
Costs	Direct costs are low. Business consequences of delay very costly	High capital cost, but predictable maintenance costs. Overall low.	Generally high due to high markups of consulting firms.	Low	Low
Fit and flexibility	Generally poor because of time pressure, budget pressure and staffing problems	Generally adequate. Most firms cannot take advantage of all features in packages.	Fit is often good, but long term architecture likely questionable	Generally high	Not very important as likely a commodity category or service
Total cost of ownership	Very high	Low to medium	Probably very high due to poor architecture.	Very low	Low
Rate of innovation	Low	Good with successful vendors. Poor with bad vendors	None, without taking ownership of project or hiring consultants again	Very high	Low
Effectiveness	Low to medium	Generally high	Low unless there are no other alternatives	High	Medium to high
Critical success factors	Great staff, modern OO development approach	Picking the right supplier with the right mix of features, costs and ability to innovate	Good staff, good architecture, design for flexibility, transfer of knowledge	A vendor that will survive and grow. Vendor ability to retain staff.	Commodity functionality, supplier must operate effectively and retain staff

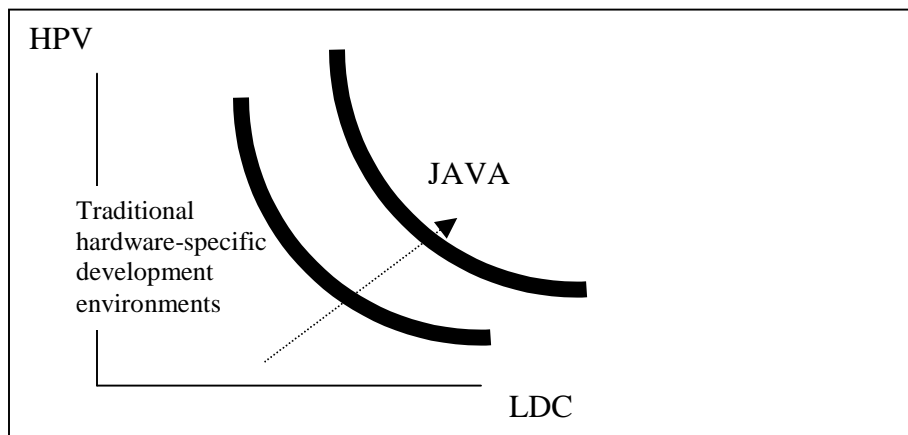
If we narrow our focus to looking at a particular area, generally speaking, competitors can be categorized using the HPV and LDC grid. HPV stands for High Perceived Value and LDC stands for Lowest Delivered Costs.

Turn Around!



In rough strategic terms, the functionality offered by Microsoft and Sun are approximately the same. They are merely positioning their products at different price points on a marketing curve.

In contrast, JAVA is an attempt by Sun to compete with development tools from established vendors such as Microsoft, IBM, and Symantec by offering a superior value proposition – portability.



In contrast, the launch of Java is strategically more significant as it changes the economics of delivery and deployment. Sun altered the relationship between perceived value and total cost of delivery.

In brief, these two examples highlight the difference between strategy and marketing. The difference between MS-Office and Lotus SmartSuite is a marketing difference. The

Turn Around!

success of the two companies' products is not based upon any fundamental difference in capabilities.

In contrast, the launch of JAVA by Sun changes the rules of the game and trade-offs customer can now make between value and cost. They have created a new curve with different price-value positionings.

The strength of the JAVA strategy is reflected by its adoption as a standard by all major tool vendors. IBM/Lotus, Symantec, and Microsoft all have their JAVA offerings.

Becoming a Standard

The Holy Grail for all software vendors is to define and own a standard. The examples are numerous:

- MS-Windows
- MS-Office
- Adobe pdf format
- Lotus 1-2-3 in the eighties and nineties
- IBM SQL in the early days of relational databases
- Ashton-Tate's DB2 in the early days of PC databases
- Autodesk's Autocad
- Sun JAVA and EJB

Achieving the position of dominance and premium pricing that a standard definer obtains generally is triggered by the following situation:

1. A visionary firm that defines a semi-open architecture, interface or tool.
2. A space in which existing solutions are significantly inferior to the new technology.
3. A rapidly growing need for solutions in the space.

However, the situation is not enough. The management of the software firm must:

1. Actively pursue a semi-open standard setting role by opening up its technology sufficiently to allow other firms to make money off it. Failure to do so causes early winners like Apple to fail to create the winning standard.
2. Encourage the development of channels for promoting partner solutions. These channels have in the past include directories, magazines, training purveyors, etc.
3. Branding of third party solutions by providing certification.

Semi-Open vs. Open

I am in favor of semi-open solutions for three important reasons.

First, in the early stages of development of a standard, the key individual or individuals that define the standard can move much more quickly than committees.

Turn Around!

Second, many firms will join committees for developing standards merely as a way of slowing down the process or bending it towards their strategic interests.

Third, it is more profitable to control the standard and evolve it towards a vision than to let its focus be dispersed by multiple stakeholders.

Now, some will disagree with this conclusion. Eric Raymond, author of *The Cathedral and the Bazaar* has some strong arguments for the open source movement. But I suspect that open source will tend to be more practical in markets where there is a dominant vendors such as Microsoft's role with Windows and MS-Office, where LINUX, Netscape's browser and Star Office are explicitly positioned against a dominant standard. It may be harder for specialty products and services to attract the interest of generalists.

Pricing Software on the Internet

If you are an ASP, one of the frequent concerns is, "How should I price my software?"

The first answer is that there is no single answer. Different pricing schemes will produce different rates of customer adoption. One of your tasks therefore, is to test your pricing and see which produces the best sign up yields and which produces the best customer relationship multi-period profitability.

In one ASP company I worked with, the challenge was first of all to figure out who the competition was. The management had been fixated upon the one visible competitor in their space. Their pricing was based upon an up front fee, a transaction fee and a percent of value fee. They also had a maintenance fees. For their mid-market clients, they would sell the software. For larger clients, they would not sell their software, as they did not want to give up their transaction revenues.

They had forgotten, that in coming up with prices to test, you need to figure out the economics both of your business and of your customer's usage. Because competition is only one variable, the other variables are (1) what is the total cost of ownership or TCO for the prospect, and (2) what is the business consequence of being able to be faster to market.

Turn Around!

Chapter 7: Improving Sales Performance

Having done a lot of business development, sales and evangelism work for companies that I have started, I have observed that managing sales costs and sales cycles is one of the glaring weaknesses of many companies. It seems that fixing the business model of most companies typically requires changing the sales strategy for the firm.

There are, I believe, 12 key conclusions about the economics of selling high tech products.

1. There is a limited window in which you can sell your high tech product.
2. Segmentation choices affect the length of your sales cycle. Testing sales cycles and focusing upon segments with rapid sales cycles dramatically improves financial and sales performance.
3. The high tech sales cycle typically has 6 components.
4. There is a clear trade-off between short-term usage goals and relationship profitability.
5. Rapid evolution of high tech products improves product performance and customer satisfaction.
6. Rapid evolution requires improved measurement systems.
7. Product development is about both product and services development.
8. Distribution and strategic alliance strategies are a source of costs that are typically underestimated by companies.
9. Highly leveraged business models and market leadership are not restricted to technology-based leadership strategies.
10. CRM (customer relationship management) requirements are rarely planned for and are a significant component of the selling process.
11. Measurement of the impact of price, feature sets and different value propositions is rarely attempted.
12. Market research can help far more than most firms realize.

Turn Around!

Companies that understand these twelve issues can frequently improve their performance in a short period of time. Companies that don't understand these issues tend to continue to experience deteriorating financial and marketing performance and eventually go out of business or are acquired.

So, let's examine these conclusions one at a time.

There is a limited window in which you can sell your high tech product.

Almost by definition, a high tech product is one with a short life cycle. While there do exist some high tech industries such as pharmaceuticals where patent protection does provide some protection and barriers to entry do exist, most high tech firms have to deal with rapid life cycles and a need for rapid learning. In the computer laptop business, product life cycles run about 6 months. In the software business, product upgrades are frequent. Annual upgrades are common. In Internet businesses, product improvement is frequently continual.

What this translates to is that there is optimal period in which sales activities are most productive and where the value of the product is at its highest. This short window requires that you manage your sales cycle to obtain sales during that window. Otherwise, by the time the sale is about to close, the value of the offering will have deteriorated. In some cases, this may lead to a failure to close the sale.

When a company has inadequate resources to market and sell its product, it needs to have a sense of urgency about changing its strategy. Otherwise it will miss its window of opportunity.

Segmentation choices affect the length of your sales cycle. Testing sales cycles and focusing upon segments with rapid sales cycles dramatically improves performance.

One of the hardest lessons for entrepreneurs to learn is that segmentation does count. While in theory, many high tech products or services can be sold to many different types of buyers, selecting the right segment has dramatic consequences to the cost of sales, the need for working capital and the rapidity with which a company can improve its product and grow.

Turn Around!

Take a simple example.

Assume that you have one sales person and that it takes one month of effort to close a sale. Make the simplifying assumption that his closing rate is a very high 50%, so in a year he can generate 6 sales

If instead, the company targets an industry with a 6 month sales cycle, the sales person can generate one sale per year. Under the second scenario, the size of sale must be at least 6 times larger to compensate for the smaller number of customers and a higher need for working capital to finance the selling process.

However, there are some dramatic performance benefits to having more customers.

First, the company can learn more from six customers than they can from one customer. And at the end of the year, they will have on average, one customer with 10.5 months of operating experience, one customer with 9.5 months of operating experience, one customer with 8.5 months of operating experience and so on.

If it takes 2-3 months to get useful feedback from the early customers, the company has the opportunity to improve its sales, training and support process with the early customers to improve the sales and usage experience for the later customers.

The company with the large and more infrequent sales will not have had this experience. On average, the first sale will occur in Month 9 and the benefits of learning are pushed into year 2. Flaws in the sales and support process, requests for feature improvement based upon usage and bug fixing is delayed to the detriment of the product and service quality relative to other customers.

Turn Around!

The high tech sales cycle typically has six components.

1. Time to qualify.
2. Time to close.
3. Time to competence.
4. Time to productivity.
5. Time to repeat sale.
6. Time to reputation.

Many high tech firms with leading edge products run into the problem that they confuse interest in their product with an intent to buy. Qualification of prospects ability to buy quickly is critical. Without this capability, no leading edge firm will survive.

Focusing upon only the rapidly closing customers is clearly insufficient. The more high tech products that I work with, the more obvious it becomes that many categories of high product are so complicated that selling is only the first stage in a value creating relationship.

Getting the users to actually use what has been bought is the next and equally important challenge. In areas such as software and Internet businesses, making sure that usage occurs is a critical activity. In the case of a software tool, it may require actually providing professional services to make sure that the client has a quick and early success. In other businesses, usage may require training and customer support programs.

In many high tech businesses, trial and usage may be obtainable, but it is the repeat sale that is the source of profitability. Many high tech suppliers fail to manage this next step by not making sure that they assist the client in understanding the benefits of the usage of the product. This “time to productivity” requires that the client understand that the initial usage of the product has had pay-off and there is now strong evidence for purchasing more product or services.

Practically, this assistance can mean creating cost benefit measurement models, white papers, business cases, or actually doing measurement work on the customer business. Recognition of productivity is however, not very useful unless it can be converted into a repeat sales. And the data is very clear that repeat sales are less expensive and more profitable.

The final stage in any high tech business is what I call “Time to reputation”. At a certain point, a high tech supplier has demonstrated its ability to reliably create value and service the customer. At such point, buying decisions start to be predicated far more on the business consequences of the usage and less on the more tactical issues of features and price. Reaching this stage is the objective for all product launch and sales strategies. It is at this stage in the market that competitors will start to say things such as: “We lost the deal because nobody ever got fired for buying IBM/Microsoft/Intel.” Companies like Adobe have achieved this status with their Acrobat product.

Turn Around!

There is a clear trade-off between short-term usage goals and relationship profitability.

In the late 1990s, many Internet businesses built their user base by giving away software. Some such as Netscape were fortunate enough to get bought before they monetized the relationships by converting users into paying customers. In the harsher light of the Year 2000/2001 NASDAQ meltdown, companies are now faced with a more traditional requirement of building self-funding businesses.

In sales terms, discounting or giving away software to create a standard has the enormous benefit of reducing the effort required to create a standard. However, in a world of competitive “give-aways”, the give-away is not a business, it is only a sales tactic for the profitable or optimized sale of something to be purchased later. So except in unusual market segments, putting a lot of effort into give-aways makes little sense without an upgrade strategy.

So in effect, the trade-off is the following:

1. The cost of getting the give-away into the hands of customers and the upgrade sales, marketing and CRM costs offset by the revenue from the upgrade and future multi-period revenue opportunities, versus
2. The working capital and capital investment required for a more selective sales process where the company is charging for its services.

Another way of looking at the same problem is to express it a single period profitability goal vs. a multi-period profitability goal. Clearly, in looking at the behavior of Microsoft vs. Netscape, Microsoft had the advantage of deep pockets allowing it to take a multi-period view of browser customer profitability.

For the well funded company, there is also the concept of optionality. For certain classes of products, there may be advantages to placing an initial product into a client as a way of ensuring access to subsequent or related purchases. From a probability perspective, the revenue stream for such companies would be:

Initial sales Revenue PLUS (Probability of sale on subsequent purchase TIMES revenue from subsequent sales)

More sophisticated extensions of this formula would take into account the time value money, the different probabilities of different levels of revenues and life cycle consequences.

Turn Around!

Rapid evolution of high tech products improves product performance and customer satisfaction.

Most innovation and product improvement results from customers actually using the product or service. The following table shows the dramatic impact in quality (expressed relative to a launch value set to 100%)

	Improvement per customer				
	10%	1%	0.10%	0.00100%	0.00010%
<i>Launch value rating</i>	100%	100%	100%	100%	100%
<i>Value rating after</i>					
# Customers = 1	110%	101%	100.1%	100.01%	100.0001%
10	259%	110%	101%	100%	100%
100	13,780%	270%	111%	100%	100%
1,000	n/a	20,959%	272%	101%	100%
10,000	n/a	n/a	21,916%	111%	101%
100,000	n/a	n/a	n/a	272%	111%
1,000,000	n/a	n/a	n/a	22,025%	272%
10,000,000	n/a	n/a	n/a	n/a	22,026%

In other words, in a specialty product where learning from each customer is very high (contributing in the table to a 10% improvement in value over launch value) the difference between having no customers and one customer is a ten percent improvement in value, but the difference between launch value and having ten customers is a value rating of 259% or 159% improvement.

Clearly, at a certain point in time, the ability to extract learning from customers starts to drop off and mathematically, the above model runs into an exponential growth problem. But interestingly, an offsetting view might be that with enough customers, new learnings emerge about the value requirements of a segment or on a 1-to-1 or personalized marketing basis. Businesses based upon customer intimacy can, with enough purchase frequency, experience a similar value enhancement on a per customer basis.

The same effect holds true for mass markets where each new customer contributes proportionately less learning about overall value creation. What is particular interesting here is the mathematics of value improvement – for product categories that require extensive customer learning or where customer needs are evolving quickly, rapid learning can lead to disproportionate creation of value if enough customers are acquired quickly enough. And if capital investment is required in e.g. data mining or customization software, size may be a requirement for affording the analytical and delivery capabilities.

Now, some readers may feel uncomfortable with this analysis, but if we think about the change in value perception of customers, we are not talking merely about features improvement or bug fixes. Value to a customer covers many different areas – e.g.:

- the benefits of a using a product that is becoming a standard

Turn Around!

- the ability to exchange information with other users and customers
- the existence of knowledge, support, and magazines focused on the high tech product
- the availability of complementary products
- certainty about the survival of the product due to market share and the ability to provide support
- the ability to personalize products and services based upon customer intimacy.

The ability to increase value and stay ahead of the competition is a key issue in sales, because as every sales person knows, a superior value proposition is easier to sell. And the data on new product development is unequivocal – a high level of differentiated value causes a 98% likelihood of product launch success. In other words, higher sales.

Rapid evolution requires improved performance measurement systems.

Having demonstrated that learning from customers can help companies improve their value proposition in terms of products/service performance, segment value or personalized value, the question arises: “How does a company capture these opportunity for improvement?”

There are no simple answers to this question. Every company is going to be different and with varying resources prioritization will be critical in selecting the methods. used. Approaches will include the following:

1. Implementing a [Balanced Scorecard](#) approach to organizational measurement.
2. Performing frequent customer satisfaction research.
3. Pursuing informal ways of getting to know the details of customer businesses and usage, e.g. living with the customer for a week.
4. Standard market research that focuses not just upon what the customer says but what the customer does.
5. Improved employee training.
6. Improved sales force reporting systems.
7. Improved usability and performance tracking measurement at points of contact with customers.
8. Creating a culture in the organization that puts the customer first in decision-making processes.

Product development is about both product and services development.

One of the most common mistakes made by high tech firms today, particularly the ones I have consulted to in Silicon Valley is to focus only on the technical side of their business. One company with whom I worked was so obsessed by the building of a piece of software that it did not notice for 8-months that it had been walking away from major service revenues that would have obviated the need for fund raising – a big plus in a difficult funding environment. And they are not alone in this blind spot.

Turn Around!

The fact is that engineers and programmers like to build things. And having authored a number of high tech products, I certainly understand the attractions of creation and enjoy creating and building products. But customers want solutions. And often they do not care about the mix of the solution – whether it is all self-service or whether it is a mix of product and services.

By missing this simple fact, many high tech firms increase their difficulty of raising money. By changing their sales process, they have the ability to solve multiple problems at the same time.

- First, sales cycles can be shorter.
- Second, they can gain a more intimate understanding of customer needs and perceptions of value.
- Third, they can reduce their need for fund raising.

Highly leveraged business models are not restricted to technology-based strategies.

Another common misconception in the Valley, prevalent among venture capitalists and those companies seeking venture capital is that the only leverageable business model is a “high R&D” software or hardware business. The logic goes as follows. In software for example, margins typically run very high. Depending how you allocate costs, software has a margin of 85-99%, so each incremental sale contributes highly to cash flow.

However, what this analysis forgets is that R&D and marketing expenses must be allocated over the volume of the product sold. And if your sales cycles are too long, you will not have enough time to close the required number of sales to amortize your fixed costs. So in effect, you will not make money on *any* sales.

So the apparently high margins of many high tech businesses will only materialize in the event of success. And success is difficult in market where multiple competitors are being funded by multiple venture capital firms, or where marketing resources are weak because funding has dried up.

For such reasons, I would argue that companies that take a less theoretical view of solving customer problems are more likely to be self funding, more likely to develop good knowledge of their customers and be more likely to succeed.

As Treacy and Wiersema point out in their book, [The Discipline of Market Leaders](#), there are three ways of seeking leadership:

1. Technology leadership,
2. Process leadership, and
3. Customer intimacy leadership.

Turn Around!

Technology leadership is the default bias in Silicon Valley. However, when I point out to clients that Dell is arguably the most successful personal computer company ever, they realize that process leadership can build a huge business.

In the future, it is my belief that customer intimacy leadership will be perceived as increasingly attractive. The explosion in technology has caused indigestion in many target prospect companies. They cannot keep up with all the technologies required for their business. So a straight technology sell will in many cases be inferior to a services, ASP, out-tasking or solution sell.

Customer intimacy not only translates into improved product and service, but also a better ability to target the right product and also increases the probability of repeat purchase.

**Example: Margins Increase From 50% to 76%
as % of Repeat Sales Volume Increases.**

Mix of customers	Cost of New Sales Efforts	Cost of Repeat Sale Efforts	Weighted Contribution	Weighted Contribution	
Repeat purchase %	As % Sales	As % Sales	From New Sales	From Repeat Sales	Total Margin
0%	50%	15%	50%	0%	50%
25%	50%	15%	38%	21%	59%
50%	50%	15%	25%	43%	68%
75%	50%	15%	13%	64%	76%

And clearly the more that customer education is required, or long approval processes are experienced, the less profitable pioneering sales efforts will be. Another implication of these figures is that many high tech firms underestimate the synergy of selling additional and complementary products and services to existing customer bases. In a customer intimacy leadership strategy, owning the relationship allows a different procurement strategy for resale.

Managing the relationship with existing users and customers leads rather naturally into the next issue of CRM, because existing customers are more likely to visit and use lower cost forms of interaction with a high tech firm. But identifying when the high cost human interactions are necessary, is a critical activity in maintaining customer intimacy leadership.

CRM requirements are rarely planned for and are a significant component of the selling process.

Another set of costs that is typically ignored in sales strategies is the cost of developing the customer relationship management or CRM process. Companies typically discover that if you analyze the cost of CRM prior to launching products and services, it changes the profitability of proposed new products. Working with [Primary Matters, Inc.](#) to model

Turn Around!

this problem for one start-up, we found that the company would need 300 support staff by the end of year three because of the variety of services that the initial founders and sought to provide. With better costing information, the strategy had to change.

Another misconception about CRM costs is that they are costs. For many companies, the variety of CRM interactions actually represent part of the sales force, have different cost structures than expensive direct sales activities and need to be modeled as a revenue- rather than a cost-center.

Distribution and strategic alliance strategies are a source of costs that are typically underestimated by companies.

Many small innovative high tech companies believe that the cost of a direct sales force is prohibitive. They turn instead to indirect sales and strategic alliances. Perhaps the most important lesson in negotiating and developing such distribution strategies is that many inexperienced startups do not anticipate that the sales and sales support costs for such relationships are exceptionally high. I have frequently seen small companies accept distribution deals, development contracts or alliance deals with large firms but not be able to exploit such relationships because they cannot manage the large investment needed to make the typically large alliance partner effective.

Again, these costs and opportunities can and should be modeled in advance.

Measurement of the impact of price, feature sets and different value propositions is rarely attempted.

In the middle 90s, I spent a great deal of time training telecom managements to deal with competitive environments. There were several consistent errors made in the microworld or simulation we created, that I see time and again with high tech firms – that is, most managers are reluctant to use price as a marketing weapon or to regulate demand for their product if they have bottlenecks or capacity constraints.

Given the emphasis upon short life cycles and unpredictable buying patterns for novel or disruptive technologies, measuring the impact of price (and other aspects of the marketing mix) takes on new importance. In many cases, management teams have been overly influenced by strategic theory and their vision of the company and insist upon excluding revenue opportunities from segments of the market that are not in “their mental model”.

Another frequent error is to fail to consider different pricing models. When you have a bundled product and service, you have the choice of:

1. Charging for the product and giving away the service.
2. Charging for the service and giving away the product.
3. Charging for both.

From a sales and customer purchase perspective, these approaches have different effects upon the closing cycle. So experimentation makes sense. If you sell your hardware as a

Turn Around!

service, it may be purchasable immediately. If it is a capital expense, it may be subject to different rules, pools of capital and buying processes.

Market research can help far more than most firms realize.

The data on new product development success says quite clearly that early marketing homework makes a difference. And I believe this is just as true with a high tech product whose features are evolving quickly. Rapid and early crisp product definition helps everyone in the organization sell the product. Monitoring the market will evolve that specification, but market research, usability analysis, living with the customer, focus groups, and attempts to understand the worth of different features, services, levels of service can only help improve performance.

And in most markets, segmented pricing will also produce a higher rate of success. One size does not fit all.

Action Steps

The action steps that come out of these 12 prescriptions will vary based upon the situation of the high tech firm. But as a general rule, the following steps will prove useful.

1. Review the sales performance of the firm by type of customer. In particular, focus on the issue of whether you are making it simple to do business with you.
2. Review the success of different channels and individuals. What lessons can be learned?
3. Model the cost of servicing customers, distributors and strategic alliance partners. Consider either (a) refocusing efforts and dropping customers or (b) testing different approaches such as selling distributed products or targeting other segments.
4. Develop a best practices sales model and train direct and indirect (CRM) sales staff.
5. Invest in getting to know your customers better. Listen to their needs rather than telling them about your product or service.
6. Develop a comprehensive performance measurement system, perhaps along the lines of the Balanced Scorecard.
7. Review the pricing model and test alternative pricing and business model approaches.
8. Ask for referrals from satisfied customers. Reward them for both leads and repeat purchase.
9. Developed improved training materials for customers, improved white papers and cost-benefit analyses, or consider offering more professional services.
10. Set goals for closing accounts more rapidly. Spend effort to achieve such goals.
11. Link product planning to the results of market research and sales feedback.
12. Create interdisciplinary product development teams that include sales staff and customers.

Turn Around!

Chapter 8: Managing the Team

Perhaps the first and most important observation about managing a software company is that it is an immensely difficult task. I like to think of it as being similar to the process of managing an arts organization.

In a development organization, the developers want to build the best possible code. The marketers want to deliver the best possible product. There are always conflicts between the two in the same way that, in an arts organization, the artist is focused upon the artistic outcome, and the manager is focused upon the budget outcome.

The role of the CEO is to make sure the technical vision is preserved and enhanced and that the marketing/sales side of the organization generates the resources and feedback to make the vision happen.

Rule 1: Protect the Developers

Great architects and programmers need to be protected from the “time sapping” activities that organizations demand. If you think about productivity, a great developer may be 100 times more productive than an average programmer. And because such developers are typically defining architectures and frameworks for use by a second tier of developers, their rapid development has even more impact on the productivity of the second ring of programmers, who are fleshing out the framework.

In other words, you don’t want Yo Yo Ma doing the accounting for an arts organization. You want him to be practicing his cello, performing or composing.

Rule 2: Don’t Have Developers Do Marketing’s Job

It is the job of the marketing organization to define what the customers believe the product should be capable of doing, of measuring what the customers are using and of coming up with customer driven innovation and change. This information should be clearly translated into documents that developers can use to plan their architecture and features additions. It is not the job of the technical team to be marketers.

The reason I put this as the second rule is that many organizations underestimate the complexity of the issues that the development team must deal with and they burden them with marketing choices. This rule is particularly important as the company moves from the initial vision towards becoming more customer focused.

Now, I am not arguing for depriving the technical side of the organization from customer interaction or involvement. I believe very strongly, and academic research suggests, that inter-disciplinary teams build better software.

But I do believe that many management teams are too lazy or too busy to do their job properly. They do not provide proper support to the technical team.

Turn Around!

Another way of putting this rule is that while architecture rules, customers also count. Changes to the architecture that are customer driven, need to be well specified to help the developers do their job well, because they need to consider the consequences of new features. Are they incremental and can be fitted into the architecture? Or do they demand a new architecture (a platform change).

Rule 3: Talk the Talk and Walk the Walk

Many firms claim that they care about producing good software, but few actually put their money where their mouth is.

Creating great software requires taking risks. It means being prepared to throw out unsuccessful code. It means pushing the limits of the technology. It means hiring and supporting great people. It means letting programmers do great work.

I believe that if you believe that good software makes a difference, you should design your workplace accordingly. And you should run your projects differently.

For example, research on developer productivity suggests very strongly that interrupting a developer when he is deep into a task can set him back 45 minutes. My solution is to give developers offices with doors that can be shut. And just as importantly, give them signs that say, “Do not disturb!”

I have found huge success with having developers work at home. Providing their home environment does not cause interruption, many developers with whom I have worked achieve more in 8 hours at home than they do in three days at the office.

Rule 4: Measurement Matters

In a knowledge-based organization, there are always going to be conflicts. Conflicts arise from unshared information or unshared assumptions.

So measurement and communication of outcomes really does matter.

Arguments over interfaces can be settled with usability testing. Arguments over features can be settled by collecting data on support calls. Arguments about marketing effectiveness can be settled by doing split run direct mail tests. Arguments over customer needs can be settled by frequent product launches and measurement of results.

In terms of programmer productivity, I am not in favor of measuring crude measures such lines of code produced or function point counting. But I am in favor of tracking how time is spent in an organization on development. By doing so in one of my firms, we noticed the following trends:

- We were not getting adequate reuse on our projects.
- We were spending 50% of our time on the database component of our projects.
- We always seemed to have the worst debugging problems at the end of the project with incompatibilities between the database and the application.

Turn Around!

There were a number of potential solutions to this problem. Better documentation, better training, better developers, better tools, or using design tools.

We decided to tackle the problem straight on. We move to a more integrated generation of object oriented development tools, pursued more training, upped the level of expertise of the team and switched from relational databases to object databases.

Over a three year period, our performance improved by 20 times. But perhaps even more importantly, we stopped finding bugs right at the end of the project where there were incompatibilities between the code and the database. We had eliminated the impedance mismatch between the object oriented applications and the relational database. Now, when we changed code in the objects it was automatically changed in the database.

In summary, measurement does pay-off, but not measurement of how long it takes to write a line of code. Again, the objective of moving up the software maturity capabilities model is a key objective for a software company building a sustainable competitive advantage.

Rule 5: Project Management Matters

Project management is a basic tool in any software company. So it deserves attention. There are some important guidelines for project management in a startup.

First, project management is a whole lot easier in the R&D phase when you don't have any customers.

Second, the moment that non-technical types are involved, arguments are going to arise. For companies that have not brought products to market before, the management teams often underestimate the amount of work that it takes to productize and release a product. Normal steps include:

- Alpha testing i.e. in-house testing, preferably by someone different than the developer.
- Beta testing with customers, which typically generates a support requirement.
- Documentation, tutorial and training materials. Typically tools used for manuals can also be used for generating the on-line help system.
- Wizards for automating tasks for clients.
- Packaging, art work, printing, icon and logo design.
- Development of marketing literature.
- Training of support staff.
- Development of evangelical material.
- Development of launch, marketing and PR campaign.
- Sales training materials.

Turn Around!

Rule 6: Sequencing Matters

Most of the more confused software companies that I have worked with fail or run the risk of failing because they do things out of order. In the past several years, I have worked with clients who prior to my arrival have done the following:

1. Gone through three generations of product development without every considering the need to assemble the right technical team or have a decent product specification.
2. A company that developed its product first and then asked if anyone would use it.
3. A company that wanted to develop a family of software that was already available in the market from several different vendors.
4. Companies that have attempted to market non-existent undocumented proposed products.
5. Companies that expect large well-managed prospects to buy from small under-funded poorly managed software companies whose product was a generation one level of technology or whose technology was unproven and product incomplete.

The way of avoiding these problems is to use formally or informally the following filtering process:

1. Document your idea. Test it out on people to see whether it makes sense. Don't talk to your friends. Talk to hard nosed people. Would they write a check for this product today?
2. Do a preliminary study? Talk to customers? Live with them for a while. See if you idea is practical in the usage situation you envision.
3. Do a preliminary plan. Cost out the development and launch costs. Do a detailed analysis of the competition?
4. Build a specification. Make sure that the functionality is phased. See if you can sign up a customer based upon the specification. Be modest in your goals and propose to deliver the minimum useful feature set.
5. Build the minimum useful feature set. Live with the customer while they use it. See what can be learned.
6. Modify your specification and extend the functionality. Let customer request influence you. Try to focus on the needs of people writing you checks. And if you have too many people writing you checks, focus on the needs of the good customers with the long view.

Rule 7: Turnover is Expensive

I have a prejudice. I believe that turnover in a company is very expensive. Turnover is particularly expensive in technical teams, but it is expensive in all roles. Great companies, in my view, have low turnover. They offer an attractive working environment that allows people to be creative work hard, and have a personal life.

I cannot emphasize enough the importance of balance in a company. While you can build companies by paying people more and working them very hard, such companies tend to

Turn Around!

have high turnover. And in my experience, many people who work long hours are not very productive.

The issue for companies is being smart and effective not working long hours that hurt the emotional and physical health of employees. If turnover is costly, then creating an environment of burn-out or, as Ed Yourdon calls them, death march projects, ultimately has a very high cost.

Rule 8: Treat Customers as You Would Want to Be Treated

While there are bad customers who attempt to exploit software vendors, I believe it is important to treat customers well. Great companies build loyal customers. Losing a customer not only loses a stream of revenues, but it potentially creates bad word of mouth.

Strategically, you may decide that only certain kinds of customers are profitable for you to serve. But the decision to eliminate customers should be a managed decision, not one that happens because service levels happen accidentally.

Dealing with Performance Problems in the Team

Hiring knowledge workers is never a reliable process. There is a certain magic in creating an effective team. Sometimes they work. Sometimes they don't. There are typically four basic kinds of problems:

Values and Philosophy

Great programmers build software differently than mediocre programmers. They finish what they start. They often tend to document better. And they have a much clearer idea of what they are trying to achieve.

Encouraging mediocre programmers to follow these precepts has in my experience not been hugely successful. You tend to have it or you don't.

Project allocation problems

Developers are very aware of whether they are working on "interesting" or uninteresting projects. Part of being a *hot development shop* is having enough interesting development work to go around or having the career opportunity, once you have proved yourself of doing interesting work.

I believe that it is good practice to have regular performance reviews (monthly or at the completion of a milestone, whichever comes first), because frequent feedback is the best way of improving staff. But there should be a long term plan for every developer in the team.

Turn Around!

Programming skills and aptitudes

Some programmers are better than other programmers. It's hard to tell why. Most of my best developers have been musical, but others have different talents that show up in their code. One of my favorite CTOs used to tell me he wasn't very good at remembering things so he always tried to wrap up project components and document them so he would never have to rely on his memory.

Communications skills

The kiss of death for a programmer, in my experience, is the inability to communicate. Even worse is the inability to admit that you don't know how to do something. Programmers who cannot admit their ignorance cannot grow. Not being able to understand direction also makes for short job tenure.

Communicating well is a key task for a team leader and often the ability to work the whiteboard is particularly important.

Performance problems are inevitable on any development team. The challenge is always to catch them early, but in small companies, one person failing can change the critical path and delay the project significantly. Even worse, remedying the problem by putting a better person on the project can have significant scheduling implications.

So the bottom line is keep the milestones and deliverables short. Formalize a code review process. Separate testing from development. Provide frequent feedback to team members.

And more generally, develop a skills development and career plan for each team member so that each one feels he is increasing his intellectual capital by being a member of the team.

Turn Around!

Chapter 9: The Internet

Perhaps one of the most important things to remember about the Internet is that it does not change the fundamental task of software development whether you are a software company selling a tool or an Internet company for whom software is a way of manufacturing a service.

Companies developing software for the Internet still make the same mistakes that software companies have made for years. They still fail to specify their projects properly. They still fail to do usability testing. They still make poor tool choices. They still get their pricing and business model wrongs.

What the Internet does do, in a sense, is give you more room to make mistakes. In the glory days of the late 1990s when it seemed as if every hare-brained idea could get funding, lots of people confused the ability to build a piece of software with a successful business.

The cynics would say that the Internet software developers made the same mistakes that personal computer software developers had made, who made the same mistakes that mini-computer software vendors had made who made the same mistakes that mainframe software developers had made.

But the major difference with the Internet comes from a fundamental law of economic geography.

“Small markets support few niches.”

The corollary of this law is that:

“Large markets support many niches.”

The Internet is in a distribution sense, the largest market in the world. It allows highly specialized companies to economically reach thin markets that would otherwise not be sustainable.

The consequence of this sustainability of thin niches is that software companies are exposed to more competition than they could ever have dreamed of before. Software is one of those products that is relatively easily sold over the Internet.

Turn Around!

The Economics of Software and the Internet

Software is a big field, but I think there are some generalizations that you can make about software economics.

First, fundamentally, the ability of software companies to reach more customers around the world means that segmentation is increasingly more refined. What might have been a legitimate software business strategy prior to the explosion in the Internet may no longer be defensible.

Second, achieving market leadership in your category is increasingly important to economic success. If you are not the category leader that everyone points to, you will not be found by search engines, reviewed in magazines or discussed in chat groups.

Third, in some markets, the software itself may be less important than the supporting infrastructure. If for example, you are a tool vendor, having local consultants, getting written up in magazines, having multi-time zone customer support capabilities start to become increasingly important.

Fourth, focus is increasingly important. While you may need customer support capabilities, you may not need to own them. Software companies always have limited capital resources. Making sure that you own core activities is one essential part of success. However, evaluating and selecting who to outsource to is now a critical activity and one that requires management. RFIs (requests for information) and RFPs (requests for proposals), often thought of as a big business skills are now tasks practically everyone had to deal with.

Fifth, the economics of software development have two components.

- The learning curve that you descend in order to solve the problem first for one customer and then to generalize about it.
- The number of customers over which you can allocate your development costs.

In general, software should be a scale game, where vendors with more customers have the ability to allocate costs over a larger number of customers and hence have higher profitability and more marketing resources.

But talent does make a difference. There is wide variation in the quality of the architectures developed by firms and the rate at which they can learn from their customers. In the long run, the truly successful vendor must learn more, learn faster, market more and gain more customers.

Learning from customers means learning *what they will pay for*. For a tool vendor like Oracle, it could be the raw tool, or it could be the value added consulting services. Or it can be services with higher value added such as applications.

Turn Around!

Noise on the Internet

Another impact of the Internet is to raise the noise level so that your marketing message is easily drowned out.

Not only is it hard to get heard on the Internet, in a world of exploding software availability, making the case for paying attention to your software is as much a problem as getting the prospect to download trial versions and actually use them.

There are two extreme reactions to the noise problem. One is to give up and go the open source or shareware route. The other is to invest in branding and marketing. My bias is towards the branding and marketing route.

Over the years, I have found that the costs of sales, marketing and support tend not be hugely influenced by the cost of the software product. Selling a \$2,000 product seems to be as much work as selling a \$20,000 product. And there is really not a huge difference between selling a \$20,000 product and a \$200,000 product. So, unless you are in the impulse purchase price range, typically under \$100, you might as well focus your efforts on larger more profitable relationship opportunities.

I use the term “relationship opportunities” because fundamentally selling software is about creating a relationship with a customer. Any great piece of software should be used by a customer over a multi-year period. The upfront sale of software is only part of the relationship. It also includes training, support, upgrades and related products and services.

Software On the Internet

Much of my activity with Internet companies has revolved around how to price Internet based software.

In reality, the pricing issues for Internet software are not drastically different than those of more traditional software companies. However, the hype of the Internet proclaims that the Internet is new.

Delivering computer services over the Internet with an ASP strategy (applications services provider) is, to all intents and purposes the same thing as time sharing, something that is done on mainframes, minis and traditional local area networks.

Perhaps the biggest difference of the Internet is not the need for pricing differently, but rather the technical options that it opens up.

For example, the Internet makes it possible to integrate an ASP application into a product or a custom project. In many situations, this ability to integrate functionality from a wide range of vendors has enormous time to market or time to project completion advantages.

Turn Around!

What the Internet does for a software developer is to expand the palette of available tools. In the eighties and nineties, sophisticated software developers were mastering windowing environments and the necessary class libraries for building graphic user interfaces (GUIs). They were pursuing reuse with the writing of component frameworks. They were exploiting data and data exchange standards to build integrated applications.

With the Internet, the notion of reusing other people's work takes a step forward. Applications can be built with class libraries, or with purchased component frameworks, can be integrated via data exchange standards such as XML, but they can delegate part of their functionality to third party ASPs. Why write a credit card validation program when you can rent it? Why develop funds transfer capability when it is already developed as a service?

This wider palette both complicates and simplifies the job of software development. On the complication side, companies need to evaluate third party technologies – a difficult task at the best of times, and particularly difficult in the early stages of a market.

On the simplification side, you can build applications faster, make them more modular and if a vendor fails, plug in a new vendor with minimal work.

The challenge with this broader palette is actually managerial. Some management teams have difficulty because of their NIH syndrome (Not Invented Here). They believe that they need to control all the software in their offering. This fallacy is roughly akin to believing that you have to write your own windowing class libraries. Companies that invest in commodity capability are wasting their scarce capital resources. They will fail as a result, and will not be able to attract investors. Just as importantly, they will be late to market with worse functionality and, in most cases, not achieve market share.

The Functionality Dilemma

Primary Matters, Inc. (www.primarymatters.com) is a firm that specializes in customer relationship management activity and resource modeling and costing. As part of their activities, they review the performance of CRM software. CRM software is a large and growing segment in the software business. Roughly 6% of the US labor force is involved in some aspect of CRM.

In their work, they have found an interesting phenomenon. Many of the best selling CRM tools are the ones with the fewest features. When they probe to find out why customers are buying these simpler tools, they discovered that companies are so busy, in some cases, so overwhelmed, that they do not have the management or technical time to set up the more complicated tools.

The simpler tools may only provide 20% of the functionality, but they solve 80% of the problems. And in a rapidly changing area like CRM, perhaps it is not worth automating everything.

Turn Around!

This example illustrates the functionality dilemma for software companies. There are always pressures to add features. Users, who know your product, want to push the use of the product. But there is always the risk that the increased complexity of a tool can deter the first time user from being able to use the product successfully.

Solutions to the complexity dilemma often require development of wizards to help first time or less skilled users or the development of a light version and a professional version of the software that users can migrate to.

This complexity problem, I believe, is the result of two major problems in software design today that occur frequently on the Internet – the absence of interface standards and the trend towards more procedural, integrated, workflow software solutions. These are addressed in the next section.

The Grammar of Software and the Problem of Procedure

Before the Xerox Star, the Apple Lisa and Macintosh, IBM OS/2 and Microsoft Windows, most software was extremely difficult to use especially for the first time user. There were two primary reasons for the difficulty of use.

First, there were just too many menus. Learning how to use them was extremely difficult. It was a bit like learning German. In German you can nest subordinate clauses with subordinate clauses until you get lost. A sentence like

You click on the button on the screen, that with a green color and the + sign, provided that the cursor not flashing is, is marked.

would be legitimate in German.

In addition, you also had to know what you had to type in order to do something. There were no clues. So to copy a file from one directory to the root directory, you had to know that you need to type:

```
Copy *.* c:\
```

But the command would only work properly if you were in the right directory, so you actually had to move to the directory for the command to work, which meant you first had to type

```
Cd c:\dirname
```

Assuming dirname was the name of the source directory.

Alternatively, you could type

```
Copy c:\dirname\*. * c:\
```

Turn Around!

All very logical, but it required training. In a world, where there was very little software, a training curve might be acceptable.

The insight of the developers of the graphical user interface (GUI) was that if you could see it on the screen, you could operate on it with less training. Picking from a menu of choices is easier than having to know and memorize what the choices are. Proving a structured form like interaction requires less understanding of syntax.

So they developed a pretty much universal grammar that uses the syntax of O-V-A or object-verb-action.

If you could see something, you could select it. So you could use your mouse to highlight a word.

Then you could pull down a menu and see what options were available and select an appropriate named menu item like BOLD.

And then you could see the outcome. If it was wrong, there was always an UNDO button. This meant that you could experiment at low cost and learning became easier. And menus were always in the same place. The apple menu was always on the left, the File menu was second, the Edit menu was third, other menus followed that were particular to the application and the Help menu was always on the right.

This type of interaction was codified by Apple and IBM developed its CUA guidelines for development.

The problem is that not all applications are amenable to this flat O-V-A menu structure. Some tasks on a computer are procedural. They require stepping through a series of tasks in the right order. Some would call it *walking a decision tree of questions*. This type of interaction requires a new grammar, a new way of interacting with customers.

The more procedural walking through a decision tree has been addressed by Microsoft in e.g. graphing in an Excel spreadsheet with wizards, which are a procedural interaction. In my own software development experience, I like to use a split screen interaction where the left hand side of the screen shows the path of the decision tree and the right hand screen is the current interaction.

With the emergence of hyper-linked documents, first on the Xerox Star, then popularized by Apple, finally to emerge in full blossom on the Internet with HTML, developers started to use hyper-linked documents as development tools.

And all of a sudden, *standardization of interface* went out the window (no pun intended). So today, we have some limited conventions about how to develop Internet applications, but very little standardization. Usability testing is now no longer optional.

Turn Around!

With less interaction with users, a need to keep customer support costs down, usability analysis pays off very quickly through:

- Faster adoption
- Fewer support calls
- Higher customer satisfaction
- Higher rates of sales

Turn Around!

Chapter 10: Humility and Other Matters of Judgment

When I first started building software early in my career, I had a vision. I wanted to build the first-ever strategic expert system – a system that would help managers improve their strategy. The system was actually quite successful, but not as commercially successful as I would have liked.

From this experience, I learned that there is a difference between stubbornness and persistence. It's a delicate line, but one that software companies need to keep foremost in their mind.

No great software is every built without a sponsor and an architect, who see the possibility of building something new that solves a problem in a new way. But you can be right in the vision, but wrong in the execution. So it is extremely critical to measure what your customers think of your product, your people and your service.

In our case, we discovered that our software was perceived by our clients as a tool that was more valuable in the hands of a consultant. It was less valuable as a stand-alone product. Ironically, the act of building the software had made us more valuable as consultants. We knew more than we realized, and our customers valued that. So we transformed ourselves into a consulting firm.

Persistence is about measuring your customers' satisfaction and letting it guide the growth of your business.

Put the Customer First

In a fiercely competitive software market, where customers can use the Internet to shop the world, a key priority is to put the customer first. In my first ASP company where I was doing a turnaround, because they had a failed product development process, I knew we need a reference site very badly.

We were very lucky to get a bank to sign up with us at a very early stage in the project restart. At that point in time, the Euro was becoming the currency of Europe. While we were an American company, and would have preferred to do business in US dollars, I insisted that we do business in Euros because it would reduce the risk of the European bank in doing business with us. This was an unpopular move within in the software company, and probably cost us money, but it was, I believe, critical to getting a name-brand reference site. We demonstrated that we cared about the customer's needs.

In the short, putting the customer's needs first may cost you money, but the reward is customer loyalty.

Turn Around!

Know Your Negotiating Power

A frequent mistake made by small companies is to overestimate customer interest, particularly from large customers. I personally, made this mistake more times than I care to admit in starting my first business.

As an evangelist, business developer, sales person or CEO, your job is often to go out and be enthusiastic about your tool, application or service. Enthusiasm is infectious, but sales is about listening to the real story and figuring who and how to close the order.

Many large customers are filled with managers who are not on the leading edge of technology. They are frequently delighted to meet with smart young companies developing innovative software and to learn from them. But they know their corporate politics well. The software selection process in a large organization is typically weighted to make sure that many issues are taken into account. Issues such as:

- Depth of the management team
- Financial stability of the company
- Demonstrated reference sites
- Scalability of the solution
- Generation of development
- Features list
- Pricing

The result of such criteria is that large companies will generally not buy from small, unproven companies unless they have absolutely unique capabilities that are light years ahead of more established competitors. And in many cases, even if you have exceptional capabilities, they will still not buy from you.

So what can you do?

The first thing is that you have to be absolutely ruthless with large companies. They can waste so much of your time, that you are actually better off not selling software to them, but rather getting an assignment to write their request for proposal (RFP) or request for information. Of course, getting such an assignment certainly gives you an inside edge if you do choose to bid.

The second strategy for closing a big company is to wrap yourself in the flag of a more acceptable company – a systems integrator. While as an entrepreneur you may feel that these companies do not add a great deal of value, what they do add is reassurance. And large companies will often prefer to pay a premium for reassurance, or to be more cynical someone with deep pockets that they can sue if things go wrong. Even in situations where you know far more than the systems integrator, the organizational perception of many clients is that the traditional integrator can assess and assume the risk of the project.

Turn Around!

A third strategy with large companies is to sell a *proof of concept* project. In my career, I have found this to be about a 50:50 proposition. I have generally had more success with this approach when the company is smaller.

A fourth strategy will appeal in situations where the large client company knows it has no choice but to innovate and take risks. You can offer them a limited royalty where they can recover the cost of their project with a royalty on the functionality developed specifically for them. But there should be an upside. This is not a bad approach if positioned as a risk sharing approach and is combined with a non-fixed price contract: it requires that customers be sophisticated enough to understand that what it wants is risky to develop, no matter whom they choose.

For the sales person, tasked with selling to the large company, the challenge is to make sure that he discovers the following information:

1. Is there an actual project with requirements?
2. Who is the decision maker? Does he have real power? Is he committed to the project?
3. What is the internal justification for the project? How does the business case look for our solution?
4. Who are the influencers on the decision maker?
5. Who will be the users of the project?
6. Has the budget been approved?
7. If it has not been approved, what is the budget cycle for the project?
8. Even if approved, what factors will influence the start of the project?
9. Is the prospect competent to successfully start, deliver and deploy the project?

If any of these are negative, then the sales person must minimize his investment in selling to the firm or refuse to sell to the firm. One of my most successful sales tactics in the past has been to say to large firms. “Look, I can’t afford the time it will take to sell to your organization. How can you help me shorten the process?”

Unless a firm has an exceptionally capable product, developing a sponsor in the large organization who sees you as his ally is likely the only way for a small firm successfully selling into the large conservative bureaucratic organization.

A Useful Sales Framework

One of my distributors used to use the following sales framework. There are three kinds of customers in the world he would say:

Turn Around!

Category 1:

Those who know they have got a problem and have some idea of what the solution is.

Category 2:

Those who know they have got a problem and don't know how to solve it.

Category 3:

Those who don't know they have a problem, and so would not be interested in a solution if it fell from heaven.

With limited sales resources, you have to focus your sales efforts on Category 1 customers. If a Category 2 customer approaches you, you *can* sell to them. But, under no circumstances should you bother to sell to Category 3 customers.

What is problematic for the sales person is that within a Category 3 company, there may well be a Category 1 or 2 manager, but unless he has a great deal of authority, the Category 3 company will be unlikely to buy.

Turn Around!

Chapter 11: Managing For Value

While IPO valuations of software and Internet businesses have been all over the map in the period of 1998-2001, there are some traditional ways of valuing all businesses that can help determine the value of any software/Internet business. However, such models need to be tailored to the unusual characteristics of the software/Internet businesses.

The basic theory of company valuation states that a company value is driven by its free operating cash flow. Free operating cash flow represents the cash that is spun off after reinvestment necessary to sustain the business. Discounting the *free operating cash flow after reinvestment to sustain the business (FCF)* is the major component of typical valuation models used by financial analysts. (Clearly, adjustments can additionally be made for initial cash position, risk, inflation, cost of capital and residual value creation from IPOs or acquisitions.)

FCF is not a number that is normally reported by the financial staff in a software company. At the early stages of a company, the focus is upon product completion, launch and reaching a positive cash flow. But a positive cash flow in the software business is not necessarily a sign of good health. If stable sales require continuing investment in R&D just to maintain the state of the business relative to competitors, then the FCF may be negative.

Choosing what to compare yourself against in calculating FCF – current revenue levels, market share is a strategic decision driven by the values, vision and mission of the company. But for the sake of convenience, let's refer to the state as the Steady State for the company.

Measuring Free Cash Flow

In my view, the Steady State for the company will in most cases be driven by the stage in the market life cycle for the company's products. To simplify the analysis, I propose to focus on the single product company.

For the early stage company, where buyers are leading edge adopters, the strategic objectives are typically to meet the specialized needs of leading edge adopters in order to build a productized product that will be acceptable to the mass market as the product category emerges and the market switches to a growth market.

For the growth market stage company, the strategic goals are typically to build a brand so that in an uncertain world, buyers will select the software over less appropriately or well branded software. During this period of rapid revenue growth, the objective is to drive down the cost of delivery while improving product/service performance and enhancing the brand.

Turn Around!

In the mature phase of the market, the strategic goal is typically to gain a low cost position with a superior value product and high market share, reaping the benefits of the learning curve descended, and building a scale barrier to entry.

At each stage, the FCF Steady State will be different. If you are not investing sufficiently in R&D, working capital, staffing and infrastructure to maintain what we might call the Strategic Steady State, then cash flow will overestimate the value of the business by taking into account the need for transforming one Steady State into another.

In other words, the Steady State for an early stage market will not normally create a sustainable position of competitive advantage without additional investment.

Defining a Sustainable Business

Clearly, in setting the objectives for each stage of the business, strategic judgments have to be made about:

1. The length of the phase.
2. The size of the market.
3. The level of competition and customer demand with its resulting impact upon pricing.
4. Estimates of required marketing, sales and support costs to obtain incremental cash flow at varying different levels of performance.
5. Expenditure levels to achieve performance measures for a basket of key performance measures grouped for example in categories such as finance, customers, operations and employee capabilities.

Dealing with Missing Information

Clearly, brand new companies will practically always have difficulty valuing themselves. Without customers and a sales history, they will have greater uncertainty about the return on sales and marketing investment. Market size may also be difficult to estimate without customer validation. However, if analogous data on similar businesses with similar market structures and value chains exist, then even these early stage companies can build a prospective value and investment model.

However, for companies with enough data to understand their sales and marketing costs, estimates can be made more easily.

Modeling Value Creation

Even within a single product company, there exists a portfolio of investment choices. Every company has a value chain consisting of activities. Activities are done on behalf of a segment targeted by the company. Each activity can be driven by six major classes of cost drivers:

Turn Around!

1. Learning or experience curves.
2. Scale cost drivers.
3. Capacity utilization cost impacts.
4. Complexity and focus effects.
5. Yield and quality drivers.
6. Time and coordination cost drivers.

Senior executives play a key role in software/Internet companies by choosing which activities and which cost driver improvement merit investment by the company. And of course, the importance of cost drivers will vary as the market matures.

If the company is targeting two different segments, then the analysis becomes more complex. There are two value chains supporting different segments with differing perceptions of value, but shared activities. The decision on how to allocate costs and relationship between the two segments (e.g. value added resellers and end-user customers) can become complex.

In some cases, mergers and acquisitions are useful tools for acquiring expertise, knowledge, improved processes or changing the cost drivers of a business.

As mentioned previously, one of the challenges of software company cash flow forecasting occurs over multiple period forecasts. Because good businesses, whether software or Internet, must rely upon repeat sales and customer multi-period profitability, the major uncertainty for companies is ensuring that they are paying attention to retention. Focusing upon customer acquisition cost, without putting in place programs for measuring customer satisfaction will inevitably cause customer defections. Judgment about the value of a company is often predicated upon assumptions about customer retention rate. Amazon.com is an obvious example. It will fail if it cannot make multi-period relationships profitable and its value will rise or fall on its success in managing customer relationships and repeat purchase.

In early stage companies, learning and experience curves are typically very important.

- Technical developers are learning how to build an effective product and begin the process of engineering the software development process.
- A new management team is learning how to work together, make decisions and put in processes that are appropriate for the company's current complexity.
- Determining how to hire, train, motivate and manage staff that are a fit with the company's mission, values and value add requires experimentation.
- Marketing is experimenting with different marketing programs, pricing schemes and distribution approaches.
- Sales staff are experimenting with different classes of prospects.
- Customer support is learning where the users are having problems and where costs are being incurred.

Turn Around!

In growth stage companies, additional elements come into play. Some of the lessons from the early stage market need to be unlearned to “cross the chasm” and become successful with the mass market. New areas of focus now include:

- Growing the scale of business.
- Changing the culture of the business towards a more marketing, test oriented, empirical culture that relies less on intuitions.
- Internationalization of the product.
- Launching the product in new market segments.
- Launching the product in new countries.
- Increasing the breadth of the functionality or services.
- Adding additional value to the product or service offering.
- Developing new channels of distribution, strategic alliances, major account sales and installed base marketing programs.
- Improving business processes and reengineering work-flows.
- Development of improved infrastructure for decision-making, budgeting, planning and control.

In the growth stage, companies need to ensure that the complexity of their business has not rise too high. Complexity generally leads to higher costs and a key objective at this point in the market is to keep improving the cost structure of the firm. Firms that become too complex such as IBM had to go through a process of shedding 90% of its business lines in order to be able to focus on the growth areas of its business.

Issues such as yield, quality management, ISO 9000, out-tasking, supplier selection and software engineering become increasingly important at this stage as the expenditures become very large.

In the mature phase of the market, a certain rigidity typically sets in and the company frequently has difficulty simultaneously driving down its cost structure, and simultaneously investing in the next generation or family of innovative products. Organizational restructuring is common at this stage to ensure that over- or under-investment in old and new products respectively does not occur.

Managing and Measuring Initiatives

Some managers may feel overwhelmed by the theoretical and financial framework outline above.

The simplification of all the valuation models is that companies in rapidly changing industries such as the Internet or software need to manage themselves based upon:

1. Dividing themselves into business units (or perhaps product or segment lines).
2. Within each business, evaluating the return on the initiatives associated with each business. This evaluation mode, in effect, demands that incremental claims for

Turn Around!

- resources need to be justified in terms of predicted improvement in financial, marketing, operations and employee performance measures.
3. Selectively investing in cross-business unit initiatives that create synergy across the business units.

This *initiative view* of a company is analogous to the capital budgeting process, the gate point new product processes and financial approaches to measuring return on invested capital.

However, because looking at projects in isolation has historically proven to be ineffective, it is important to look at return on capital employed (FCF divided by capital used by the business) only at the business unit level.

Defining a business unit is a subject of much controversy. Companies pursue different organizational strategies at different points in their life. But for the purpose of simplicity, the objective of defining a business unit is to create a management team that controls a value chain. This value chain must support their ability to create a successful business, and empower the management to implement effectively without becoming involved in value subtracting activities due to organizational complexity.

Within a business unit, I would argue strongly for employment of the balanced scorecard approach to performance management, made popular by Kaplan and Norton in their writings. The balanced scorecard approach they suggest is based upon the notion that business performance is too complicated to be measured by financial measures alone.

Initiatives should be proposed and measured based upon a set of criteria that take into account:

- The mission and values of the business unit.
- The required performance measures for supporting the mission and values.
- Performance measures should include measures in four categories: customer/marketing, operations/quality, employees/capabilities and financial measures.

Pareto Portfolio View of Value Creation

Managing for value and value measurement is, like any other management activity. It needs to be constrained. You can over invest in such processes.

Strategic management in the software/Internet business, should therefore, be the development of a process that can be pushed down in the organization to help business unit managers identify the 20% of key initiatives that will account for 80% of the performance improvement. The rest can be identified at a summary level as a part of strategic management planning sessions or more formal product plan, marketing plan or strategic plan.

Turn Around!

Chapter 12: Financing the Software Business

There are lots of books these days on how to finance startups. So I will keep my comments in this area brief.

First, most venture capitalists state they would prefer to invest in a excellent management team with a mediocre idea rather than a mediocre management team with an excellent idea. However, because there are so many startups in the software and Internet areas today, you pretty much have to have *both* excellent management *and* an excellent idea.

Second, an “excellent idea” generally requires great technology people. Great technology people *do* create value in a software company. If I were a venture capitalist today, I would be looking for superstar technical talent, preferably with full life cycle experience.

Third, venture capital firms tend to talk out of both sides of their mouth. They want focus, but they want enormous markets. The reality is that this combination is rare and where this opportunity exists, many competitors will be funded by multiple venture capital firms. So, I tend to believe that you should not get too distracted by the stated criteria of a venture capital firm. Asking a venture capitalist what he likes to invest in, produces “motherhood comments”. So it is not really too helpful to a potential investee. Great new businesses are often built around breaking the rules. (Timing and luck are good too.)

Fourth, independent of the *focus* requirements of venture capitalists and other investors, if I am betting a portion of my career on a software business, I would like to see multiple spin-off opportunities from the core technologies. In fact, the research on new product development suggests very strongly that new products that are incremental investments for a firm have a higher probability of success than products that represent brand new initiatives. So if you are going to take the risk of development a software platform, don't automatically curtail the opportunities of spinning off different implementations or applications from the platform. They will (subject to the availability of funds for marketing and distribution and the management bandwidth) represent lower risk future additions to your revenue portfolio.

Fifth, the easiest money to raise for a software firm is the money from advance payment on consulting contracts. And while bootstrapping a firm does have its own set of problems, discussed earlier in this document, it is often the easiest fund raising solution. Financing your initial R&D with a consulting contract also solves that inevitable requirement of later investors to talk to your clients.

Sixth, building a great team gives you more leverage with investors. If you can't afford to hire them up front, try the strategy of signing up a management team subject to financing being raised.

As a final note, there is no substitute for great software that does its job well, that does it better than other vendors, that is backed by a strong management team, has a well funded marketing and a talented, committed technical team. If a venture capital firm does not

Turn Around!

recognize such qualities, it is their loss and you should focus on customers who actually understand what you are trying to do. As with prospects, venture capitalists can be classified into Category 1, 2 and 3. You will only ever sell successfully to a Category 1 venture capitalist, one who has some idea of the problem and some idea of the solution.

The Financing Decision

Second guessing a venture capital firm is always difficult. The following questions are certainly ones that you need to be able to address in your discussions and business plan.

1. Is there a management team? How broad and deep is the talent?
2. Does the management team have a track record of building and launching a product successfully?
3. Has the management team had at least one failure so that they have experienced the high learning that comes from a failure?
4. Has the technical team the talent and passion to build the product?
5. Has the technical team taken a product from design to launch and revision?
6. Is the need for this product large? Does the product offer exception value?
7. Can the product be built in stages?
8. What is the quality of the design and architecture? Where are the technical risks?
9. Does the proposed product offer a significant improvement in value, functionality or cost? What are the life cycle cost implications to clients? What are the business consequences to using this software to clients?
10. Can the prospects be identified easily? What selling process is assumed?
11. What is the sales cycle for the product purchase process?
12. What is the life cycle profitability of the relationship?
13. Has the market been properly segmented? How fast is it growing?
14. Has the company identified a segment it has a chance of succeeding in?
15. Has the company projected an adequate level of marketing and business development to support its revenues goals?
16. What alternative way of solving the problems exists?

Turn Around!

17. How much investment is required to generate the first sale?
18. What is the average size of sale?
19. What is the predicted cost of sale?
20. Has the company modeled and costed the expected customer support requirements?
21. What level of competition exists in this application area?
22. How is the market likely to evolve? Will there be many new entrants? Will companies integrate forward or backward into this area? Where will there be acquisitions or mergers?
23. If the company has customers, what is their satisfaction level? Do they actually use the product? Do they plan on buying more? How do they see this product relative to alternative solutions?
24. What opinion leaders have seen the product? What do they say about it?
25. Who is on the board of directors? Who is on the board of advisors? How committed are they? How are they adding value?

Turn Around!

Chapter 13: Running Out of Money

Software companies are generally founded by technically-oriented individuals. Software development is, itself, a complicated process particular in the first generation of development. As a result, most software companies fail.

The primary reason for failure is *under-capitalization*. Software companies are easy to start, but product completion, testing, launch and achievement of cash flow are typically an order of magnitude more expensive and more difficult to achieve. And practically every software company I have seen has gambled that if they could complete the product, more dollars can be raised.

The optimism with which we all start software companies is something precious and wonderful. It is entrepreneurialism at its best.

But success requires a certain ruthlessness – willingness to define hurdles early on the game that will determine whether you should proceed. Only by defining these hurdles, these criteria can you avoid the problems of foregoing income or having to mortgage your house to meet payroll.

There are no easy solutions to starting and making an under-funded business successful, but the following guidelines may help reduce the amount of loss and risk and maximize your chance of killing bad software businesses earlier. Remember that *killing off a commercial idea quickly gives you the opportunity of tackling another entrepreneurial venture sooner. You have a limited amount of time in your career, so use it wisely.*

First, it is a really good idea to not spend 100% of your available capital on product development. Software without marketing dollars to support it rarely succeeds and the few examples of explosive growth are rare and should not be assumed to be normal. The ideal rule of thumb is to spend no more than 10-20% of your capital on bringing your first product to market.

Second, it's important to remember that for most companies, the first generation of software will at best be limited in functionality and at worst totally mis-targeted and mis-architected. You will not achieve the margins you expect on your first generation product because of the learning curve you will have descended and the need for the development of the second and third generation product.

Third, selling software is generally hard. Don't believe your spreadsheet. Probably the only strong indicator of easy sales will be shown if customers are beating a path to your door because they cannot find an equivalent solution to your software product.

Fourth, if your product is innovative, you need even more cash flow to support a long educational process. So again, don't spend all your money on software development.

Turn Around!

The Magic of Success

There is a certain *magic* to businesses that work. Events seem to flow naturally into other events. The product was simple to build. Customers had a big need for the product. Adoption and demonstration of success was fast. Little customization was required. Customers moved to purchase more product quickly. If the magic is missing in your business, and everything seems to be taking too long, consider killing the business.

Good software businesses, like any good business, are highly leveraged. Small amounts of effort create large pay-off.

For this reason, I am against sweat equity. If the business cannot support paying you as founder a market salary, then do something else.

And if the magic is missing, raising money from angels, family and friends is not going to help.

If your business model appears not to be working, then the best solution is change the business model or the business itself. But the change must be done early. One of my professors in entrepreneurial studies said wisely: “Companies that get into trouble respond to the trouble by doing more of what got them into trouble in the first place.”

If You Are In Trouble...

But these insights may not be helpful if you are already in trouble. Suppose, you have built your product, or it is close to completion, and you have 3-6 months of cash flow available. The first thing that you have to do is cut back. This will be painful. It will go against all the things I have written earlier about the value of retention, the value of being an employer of choice, but if the company is going to survive, cut early rather than late.

Your objective should be to stretch the cash flow, even if means delaying product launch. Yes, it is a costly approach, but raising money is costly too and very time consuming.

Reducing your costs means getting rid of anyone but critical developers. Putting full time staff on part time contracts or eliminating them. Some managers may be prepared to work for equity for a short period of time. Office space is probably optional in this financial situation.

And it may make sense to consider strategic alliances, distribution relationships or mergers to reduce the capital necessary to bring the product to market.

But if you put together all the numbers and it won't fly, the ultimate solution is to sell the code or bring the code to the point where it can be sold. And if you can't agree on price, there are many ways of structuring a sale including royalties and earn-outs where the price is based upon the eventual commercial success of your transferred product.

Turn Around!

As a general rule, it is dangerous to get down to 6 months of cash flow. Six months of cash flow is not a lot because launching a product generally increases expenses unless you have set up a strategic alliance or distribution deal.

Six months may seem unreasonably short to many companies, but remember that fund raising for most software companies is likely to take 3 months to get investors interested and another 3 months for negotiations and legal agreements. In difficult environments it can take longer.

Turn Around!

Chapter 14: Is Your Business a Dog?

There are no absolute answers to the difficult question of whether your software or Internet business is worth putting to sleep. But the following guidelines may help you judge your business:

Rating Your Business

Factor	Assessment	Comments
	<i>Rate each answer on a ten point scale from 1-10 where 10 is best performance, 5 is adequate performance and 1 is awful or non-existent.</i>	
Management		
1. Does your CEO have the respect of the management team?	1-2-3-4-5-6-7-8-9-10	
2. Is there a management team?	1-2-3-4-5-6-7-8-9-10	
3. Does the management team work well together?	1-2-3-4-5-6-7-8-9-10	
4. Has the management team avoided becoming burned out or emotional?	1-2-3-4-5-6-7-8-9-10	
5. Has the management broad experience with this type of product/service?	1-2-3-4-5-6-7-8-9-10	
6. Has the management deep experience with this class of customer?	1-2-3-4-5-6-7-8-9-10	
7. Is there passion	1-2-3-4-5-6-7-8-9-10	

Turn Around!

in the management team for the business and product?		
		<i>TOTAL AVAILABLE MANAGEMENT POINTS 70</i>

Strategy		
1. Does the firm have an explicit strategy?	1-2-3-4-5-6-7-8-9-10	
2. Is the strategy being implemented?	1-2-3-4-5-6-7-8-9-10	
3. Is the execution of the strategy excellent?	1-2-3-4-5-6-7-8-9-10	
4. Does the firm care about quality and the customer?	1-2-3-4-5-6-7-8-9-10	
5. Does the firm project the resource consequences of proposed projects to ensure success?	1-2-3-4-5-6-7-8-9-10	
		<i>TOTAL AVAILABLE STRATEGY POINTS 50</i>

Financing		
1. How many months of burn rate can be paid for out of capital and high probability projected cash flows?	1-2-3-4-5-6-7-8-9-10	Scale: One point for each month that the company can survive to a maximum of 10. Assume a ramp up of expenses for product launch.
2. Is there an ongoing cash flow management forecasting and management	1-2-3-4-5-6-7-8-9-10	

Turn Around!

process?		
3. Are the figures on which management is basing its decisions reliable?	1-2-3-4-5-6-7-8-9-10	
4. Are the volume assumptions for success realistic?	1-2-3-4-5-6-7-8-9-10	
5. Is the working capital trend getting better?	1-2-3-4-5-6-7-8-9-10	In other words, it is taking less working capital to grow the business based upon incremental working capital per incremental dollar of sales?
6. Is the company backed by investors with deep pockets who are interested in participating in subsequent rounds of financing? Or is the company spinning off cash and is not in need of financing?	1-2-3-4-5-6-7-8-9-10	
		TOTAL AVAILABLE FINANCIAL POINTS 60

Competition		
1. Does the firm have adequate knowledge about its competition?	1-2-3-4-5-6-7-8-9-10	
2. Does the company have a superior value proposition to its major competitors or alternative solution providers?	1-2-3-4-5-6-7-8-9-10	
3. Does the company have a sufficient marketing budget to make a significant impression on its target audience?	1-2-3-4-5-6-7-8-9-10	

Turn Around!

4. Does the company have a large competitor with a higher priced offering?	1-2-3-4-5-6-7-8-9-10	Surprisingly, having a large competitor is a good predictor of success for smaller companies. It allow them to price under the large company's pricing umbrella.
		TOTAL AVAILABLE COMPETITION POINTS 40

Product		
1. Does the product work?	1-2-3-4-5-6-7-8-9-10	
2. Do buyers actually use the product?	1-2-3-4-5-6-7-8-9-10	
3. Are buyers sufficiently involved with the product that they use it frequently?	1-2-3-4-5-6-7-8-9-10	
4. Are the customers happy with the product?	1-2-3-4-5-6-7-8-9-10	
5. Are the customers delighted with the product?	1-2-3-4-5-6-7-8-9-10	
6. Is repeat purchase a consistent pattern?	1-2-3-4-5-6-7-8-9-10	
		TOTAL AVAILABLE PRODUCT POINTS 60

Marketing		
1. Is the product being sold based upon a clear benefit message?	1-2-3-4-5-6-7-8-9-10	
2. Has the company understood the segments in its market?	1-2-3-4-5-6-7-8-9-10	
3. Has the company consciously decided not to market to certain audiences?	1-2-3-4-5-6-7-8-9-10	

Turn Around!

4. Is the company seen as the number one or two player in its segment?	1-2-3-4-5-6-7-8-9-10	
5. Is the market growing in unit terms?	1-2-3-4-5-6-7-8-9-10	
6. Is the market growing in dollar terms?	1-2-3-4-5-6-7-8-9-10	
		TOTAL AVAILABLE MARKETING POINTS 60

Sales		
1. Is the sales cycles short?	1-2-3-4-5-6-7-8-9-10	
2. Is the speed of sale improving?	1-2-3-4-5-6-7-8-9-10	
3. Is the ability of a buyer to use the product improving dramatically with new releases?	1-2-3-4-5-6-7-8-9-10	
4. Are customers accepting the price level of the product?	1-2-3-4-5-6-7-8-9-10	
5. Can the product be sold by ordinary mortals rather than the founders of the company?	1-2-3-4-5-6-7-8-9-10	
6. Does the company have effective sales literature?	1-2-3-4-5-6-7-8-9-10	
7. Does the company have an effective web site?	1-2-3-4-5-6-7-8-9-10	
		TOTAL AVAILABLE SALES POINTS 70

Technical team		
1. Is the architecture robust?	1-2-3-4-5-6-7-8-9-10	
2. Is there a clear distinction between	1-2-3-4-5-6-7-8-9-10	

Turn Around!

the tasks of CTO, VP Engineering, and Chief Architect?		
3. Has the GUI been tested for usability?	1-2-3-4-5-6-7-8-9-10	
4. Is there a coherent release system in place?	1-2-3-4-5-6-7-8-9-10	
5. Is code developed in small testable chunks and reviewed frequently?	1-2-3-4-5-6-7-8-9-10	
6. Are support requirements for the product tracked by feature?	1-2-3-4-5-6-7-8-9-10	
7. Are support requirements per user declining?	1-2-3-4-5-6-7-8-9-10	
8. Are bad programmers being fired quickly?	1-2-3-4-5-6-7-8-9-10	
9. Is the company an employer of choice for developers?	1-2-3-4-5-6-7-8-9-10	
		TOTAL AVAILABLE TECHNICAL POINTS 90

TOTAL SCORE		TOTAL POSSIBLE POINTS 500
--------------------	--	----------------------------------

Evaluation of the Ratings

Range	Rating	Comment
1-100	The dog is actually dead, but, because it is still standing up, people are confused.	Bury it!
101-200	A dog	Plan for an orderly close down
201-300	A mutt	Some signs of potential, but lots of change needed
301-400	A horse	It's tough being perfect, so be happy, you've got some room for improvement. You will go the distance.
401-500	A thoroughbred	Investors are killing to invest in you. You have a chance of winning the race.

Turn Around!

Chapter 15: Turning Around Your Internet Software Business - a guide for crisis handling

So what do you do if you are the CEO or advising a problem company facing bankruptcy in the next 3-6 months?

Having advised or consulted to a number of problem software and Internet companies over the past ten years, I have some modest suggestions that may be of help to a stressed management team. I could, perhaps, call it a 12-stage program.

1. Admit that you have a problem.
2. Involve the management team.
3. Classify your activities.
4. Develop a profile of your strengths and weaknesses
5. Rank your customers.
6. Charge more.
7. Approach different types of investors.
8. Change your message.
9. Stop developing.
10. Motivate the team.
11. Keep your eye on the ball.
12. Don't flog a dead horse

Admit that you have a problem

Doing more of what got you into trouble in the first place is generally a recipe for disaster. Perhaps the single most important rule for the company in trouble is do something different. Change is the first weapon.

Let's say for example that you are in a difficult funding environment and you have little chance of raising the money necessary for turning the business around. Historically, you have charged for software and maintenance and experienced a long sales cycle. You are having difficulty completing the product because of staffing problems. One interesting alternative to consider is opening up your source code and focusing upon servicing the customers who adopt your open source code. This was the approach taken by Netscape overwhelmed by competition from Microsoft. For more on this strategy, see the book *The Cathedral and the Bazaar* by Eric Raymond, published by O'Reilly & Associates, 1999 and 2000 and also available on the Internet at <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/> .

Involve the Management Team

While it may seem counter-intuitive to involve the very people whom you may have to fire, my experience is that management teams and staff are often far more emotionally

Turn Around!

committed to a business and will do much more than a Board of Directors might expect to extend the life of a company.

There is no real point in pretending that you don't have problems. In most high tech firms, the burn rate is well known, as is the status of fund raising activities.

And companies can be saved by creative input. Who is better placed to be creative than people who know the strengths and warts of the business? Team solutions are typically better than mandated solutions and produce more commitment to change. However, sometimes a facilitator is an inexpensive way of focusing the group. A simple gap analysis exercise can take as little as half a day.

In one firm, where I was involved in doing a turnaround, I was absolutely delighted at the response of the staff to a proposal to extend the life of the firm in exchange for more equity. Admittedly, the offer was made in 1999 prior to the market melt down; but the principle remains: that employees who have been treated fairly, will typically have pride in what has been built to date.

Classify Your Activities

The standard formula for turnarounds goes as follows:

- Twenty percent of your customers account for 80-130% of your profitability. The reason that your good customers can account for more than 100% of your profitability is that you are often losing money on 80% of your customers.
- And twenty percent of your costs accounts for most of the value that you create in your business.

Strategies for improving your profitability

	20% profitable customers	80% unprofitable customers
Good activities: 20% of high value creating costs	<ol style="list-style-type: none">1. Focus on retention strategies.2. Examine opportunities for incremental sales based upon incremental profitability.	Increase prices, terminate or reduce cost of delivery to such customers
Bad activities: 80% of low value adding costs	Outsource, stop doing, or charge for.	Reduce size of this portion of your business
Killer activities: activities that subtract value from the organization	Stop	Stop

Turn Around!

An example of “killer activities” would be the employment of bad programmers who actually increase the burden on your good programmers. The first task in any turnaround is get rid of *the value-subtracting* people and activities.

Develop a Profile of Your Strengths and Weaknesses

It may sound like Management 101, but a basic exercise that you need to do on your company is a SWOT analysis. SWOT stands for Strengths Weaknesses Opportunities and Threats. Making a list of these items in a group sometimes surfaces strengths that have emerged since the business plan was written or the stock market melt down.

One company that I worked with found that its B2C business model was essentially incapable of generating revenues, so they change their strategy and transformed themselves into a tool vendor, providing their tool to business oriented sites. The success of their strategy remains to be seen, but it has more hope of commercial success, and perhaps just as importantly, more hope of convincing their initial investors to finance the transformation.

Rank Your Customers

In addition to ranking your customers for profitability, it is also a good idea to rank your customers in other ways.

- Which of your customers could be persuaded to become strategic investors because they understand and value your software and your business?
- Which of your customers might be interested in pre-paying ASP fees or switching from an ASP to a flat fee or life-time pricing arrangements/
- Which of your customers might be willing to purchase an off the shelf version of your ASP software?
- Which customers are on good terms with you and might purchase other people’s software or services by virtue of the strength of your relationships?
- Which customers would be prepared to refer you to other contacts that they might have? Consider even the option of a finder’s fee.
- Which customers are most at risk of being switched by competitors smelling blood in the water? What programs can you put in place to incent them to stay with you?
- Are their groups of customers that could be persuaded to act as a consortium to finance or advance purchase your next round of development?

Turn Around!

Charge More for Your Services.

If you have done triage on your activities and your customers, and the company still is not economic, then more action needs to be taken. It almost seems too straightforward. But the best solution to improving your cash flow is then to charge more.

If you are going out of business anyway, and you are not making money on your existing customers, and the opportunities for volume growth in three months are low, then if your product or service has value, perhaps you can make money by charging more to fewer customers.

Approach Different Types of Investors

In the current hostile venture capital environment, deluging VCs with yet another business plan for an unprofitable venture with limited track record, may not be the best use of your time.

However, contrarians do exist and some investors are not affected by the state of the stock market. To attract the contrarians, go against the conventional wisdom and explicitly indicate that you have dropped the valuation of your company. There is nothing wrong with a fire-sale if it keeps you alive.

Strategic investors are also investing for different reasons than an eventual IPO. Who strategically could benefit from closer relationships with you? Whose bricks and mortar strategy can be helped by your pre-descended learning curve?

In situations where there are multiple competitors that are in trouble, a creative investment option is to combine forces. There are no guarantees with this approach. The process of merger *itself* can produce problems, but in general, this option seems not to be considered early enough or frequently enough.

Change Your Message

Practically every software and Internet firm I have ever worked with has always had a problem talking about itself.

Most of the time, the problem is that the company does not really understand how the customer sees the world. In addition, most companies insist on talking about their features, not prospect specific benefits.

The basic problem here is that if you have spent a long time thinking about a problem, you typically have left behind the lowly customer. The customer only cares about solving the problem, not the details. Building an Internet company and supporting software is typically about the details.

Turn Around!

The other key message change is to focus on the customers that are likely to buy fastest. The three types of customers and prospects is again a useful way of prioritizing your programs and sales activities.

	Problem	Solution
Level 1 Prospects:	Recognize they have a problem.	Know what kind of solution they are looking for.
Level 2 Prospects:	Recognize they have a problem.	Don't know what the solution is.
Level 3 Prospects:	Don't know they have a problem.	Aren't looking for a solution.

Particularly, when a company is in trouble, it is essential to focus only on the Level 1 Prospects. Their sales cost will be lower and closing will be faster.

Stop Developing New Stuff

A colleague of mine used to call it the “My Baby Syndrome”. Many early stage companies get into trouble because they won't take their product to market. There is always something new to add.

So while I am a strong proponent of only bring quality to market, sometimes you just have to freeze what you have and focus on bringing in revenues.

If you are going to develop, make sure that your development team is small and filled with talent. It is far better to have a small effective development team than a large mediocre team.

Motivate the Team

Once you have gotten rid of the incompetents in the organization, the deadwood and the good people whom you just cannot afford at this stage of your life cycle, the challenge is to motivate the survivors. Sometimes that means bringing in a new person for a key role like sales or marketing.

By involving the survivors in the triage process, they will feel that the turnaround plan is something that they can commit to.

I also believe it is important to cut deeply in order to provide some security to the people that you will be retaining. You may make mistakes and cut too deeply, but it is far worse to have a Chinese torture of a 1000 cuts that demoralizes employees on a regular basis.

Keep Your Eye On the Ball

Now that you have a plan, now that you have focused on key customers, key activities and you have staffed accordingly, it's important to stay focused. Set goals. Measure and

Turn Around!

report them regularly. Don't get distracted. Update your cash flow and sales funnel at least weekly and preferably daily. Pay attention to people's mood. Surprise them with small gifts and inexpensive events.

Don't Flog A Dead Horse

The key rule in a turnaround is don't do what you did before. If you have more than one product and one of them just isn't working, then drop it. If a sales person has yet to close a sale after 6 months, drop him or her (first prudently checking that there are not closing orders that are going to descend like manna from heaven to save the company).

Of course, changing the business is no guarantee of success, so set milestones. Have a contingency plan for each of the milestones on the road to close down. Don't do things just because that's the way we've always done them.

And bring in an outsider as an advisor, even part time, someone with a fresh eye to keep you on the road to recovery and with no commitment to the past.

Flogging A Dead Horse

Dakota tribal wisdom says that when you discover you are riding a dead horse, the best strategy is to dismount. However, in organizations like governments, hospitals, large companies, school districts, etc, we often try other strategies. These can include the following:

- * *Buying a stronger whip.*
- * *Changing riders.*
- * *Saying things like "this is the way we always have ridden this horse."*
- * *Appointing a committee to study the horse.*
- * *Arranging to visit other sites to see how they ride dead horses.*
- * *Increasing the standards to ride dead horses.*
- * *Appointing a tiger team to revive the dead horse.*
- * *Creating a training session to increase our riding ability.*
- * *Comparing the state of dead horses in today's environment.*
- * *Pass a resolution declaring that "this horse is not dead."*
- * *Blaming the horse's parents.*
- * *Harnessing several dead horses together for increased speed.*
- * *Declaring that, "No horse is too dead to beat."*
- * *Providing additional funding to increase the horse's performance.*
- * *Do a study to see if contractors can ride it cheaper.*
- * *Declare the horse is "better, faster, and cheaper" dead.*
- * *Form a quality circle to find uses for dead horses.*
- * *Revisit the performance requirements for horses.*
- * *Say this horse was procured with cost as an independent variable.*
- * *Promote the dead horse to a supervisory position.*

Source: <http://www.laughlists.com/lists6/deadhorse.htm>

Turn Around!

Decision Making Under Stress

People often say that entrepreneurialism is far more stressful than they ever anticipated. The highs are higher and the lows are lower, I often say.

It took me a number of years to recognize that one of the most difficult to manage elements for any senior manager is the stress of starting and growing an under-funded business.

Over the years, I have been in awe of the many bad decisions I have observed that were not due to incompetence, perversity or laziness, but were rather due to the fact that software/Internet businesses are almost always highly demanding. Such environments can burn out managers and staff and cause their emotions to ruin their brilliance and their ability to execute.

As a former co-author, Harvey Gellman observed, management teams often “forget their purpose”.

Turn Around!

Chapter 16: The Strategic Perspective

As a final thought, there is a simple management technique that will help all software companies through their inevitable problems and challenges. It is the process of Gap Analysis.

Often times, the stress level in companies is so high that they fail to consider all their options and they forget their objectives. With gap analysis, get the team in a room with a white board and follow the simple steps of gap analysis.

Gap Analysis

- What is our Vision and our Mission? Why do we exist? What are our values?
- What is the end state we are trying to achieve today? What are our objectives?
 - (At this stage do not get involved in the problems that prevent achieving of this state?)
- List all the barriers that prevent the achievement of objectives and the reaching of the end state.
- Brainstorm all the wild and crazy ways to overcome the barriers
- Rank and evaluate the barriers.
- Select the best actions. Have members of the team take responsibility for achieving the actions. Specify milestones and resources required.
- And of course, make sure to follow up.

I hope that this small treatise helps managers running software/Internet companies to know that there are best practices in running a software/Internet company; and that they do not need to make the mistakes that many of their predecessors and competitors have and will make.

The good news is that high tech firms continue to grow and that the experience of starting and growing a company is wonderfully valuable and it will pay off for you, whether with this venture or the next.

Good luck to you all!

Turn Around!

Appendix: More Useful Reading

Frederick Brooks, [*The Mythical Man Month and Other Essays On Software Engineering*](#), Addison-Wesley, Second Edition, 1995.

- *They Mythical Man Month* essay in this book is one of the most important essays in software engineering on the importance of small project teams.

Cooper, Robert, [*Winning at New Products*](#), Perseus, 1993

- *One of a number of good books by Cooper on increasing the probability of new product success. While focused on companies in general, it deals with the notions of stage gate new product processes, time to market issues and the importance of cross-disciplinary teams.* For assessments of proposed new products based upon his research visit www.eclicktick.com/newproduct.htm.

Davidson, Alistair, Gellman, Harvey, and Chung, Mary: [*Riding the Tiger*](#), Jist Publications 1999, HarperCollins Canada 1997.

- *The strategic selection of information management projects and best practices in project management for the information technology project.*

Kaplan, Robert, and Norton, David, [*The Balanced Scorecard: Translating Strategy into Action*](#), HBS Press, 1996.

- *A business classic that integrates the financial and strategic view of business. Two Harvard Business Review articles summarize the work for the time constrained.*

Kawasaki, Guy and Moreno, Michele: [*Rules for Revolutionaries*](#), HarperCollins, 2000

- *Good advice on innovation for Internet and software companies.*

Geoffrey Moore, [*Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers*](#), HarperBusiness 1999.

- *A simple idea. Early stage customers are different than the customers who emerge as a market begins to grow.*

Nielsen, Jakob: [*Designing Web Usability*](#), New Riders Publishing, 2000

- *Superb book on user interface design from the partner of Donald Norman, whose work such [*The Psychology of Everyday Things*](#) is also fascinating.*

Raymond, Eric: [*The Cathedral and the Bazaar*](#), O'Reilley & Associates, 1999 and 2000

- *An alternative view of software development that may be appealing to companies with products where service, consulting, content and maintenance revenues offer a viable economic strategy.*

Click on top line of Amazon Logo to obtain or get more information on the above books.



For more recommended books, visit www.eclicktick.com/eclicktickbooks.htm.

Turn Around!



The author: Alistair Davidson

Eclicktick Corporation

1560 Sand Hill Road, Suite 304

Palo, Alto, CA 94304

Phone: 650-322-8880

Fax: 707-924-2472

E-mail: alistair@eclicktick.com

Alistair Davidson is the founder and managing partner of Eclicktick Corporation, a Palo Alto based consulting firm that provide strategic, marketing and technology consulting services, contract senior managers and outsourced software development services. He consults to Internet and software businesses providing advice and is also brought in as CEO in startups and to turnarounds. He is also an advisor to TEN-Net, the San Jose/NASA Ames business incubator. He is on the board of advisors of a number of Internet and software companies.

His strategic consulting, turnaround and facilitation clients have ranged from large organizations such as AT&T, Canada Post, Baxter International to numerous smaller software and Internet companies.

He is the lead author of two prior books on technology: *Seizing the Future* (with Ralph Fisher, Trans Canada Press 1983), and *Riding the Tiger* (Davidson, Gellman and Chung, HarperCollins 1997, Jist Publications 1999).

In his first book, his co-authors and he forecast the strategic impact of technology, in particular computers and telecom convergence, in changing the shapes of institutions and markets. His second book, written with Harvey Gellman, the Dean of Canadian IT consulting, documents how best to pursue the strategic use of technology and best practices in software project management.

His career includes corporate finance with Citibank, publishing with Harlequin, venture capital with Alberta Ventures Fund, strategic consulting with two consulting firms now part of PriceWaterhouseCoopers and CapGemini Ernst & Young. In 1985, he raised venture capital to finance the development of the first managerial expert system that interviewed managers and provided advice on strategy, in addition to providing a complete set of strategic and financial planning tools.

In the middle 1990s, he led Alacrity Inc. to develop business intelligence tools that could be used for performance management, balanced score card reporting, budgeting and consolidation, project management and skills tracking using Smalltalk and Gemstone, a Smalltalk based object database. Over three generations of product development, the firm improved its software development productivity by a factor of 20X.

He has an AB and an MBA from Harvard. He was one of the early users of the ARPANET, precursor to today's Internet.