Product and Program Management: Battling the Strangler Trees of System and Social Complexity in the Software Market Jungle

by

John A. Hempe

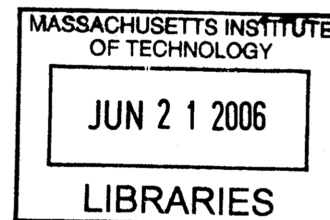
Submitted to the System Design and Management Program
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management

at the

Massachusetts Institute of Technology

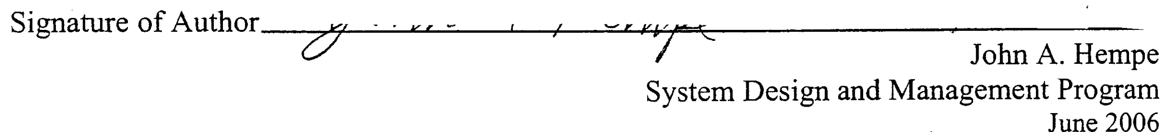
June 2006

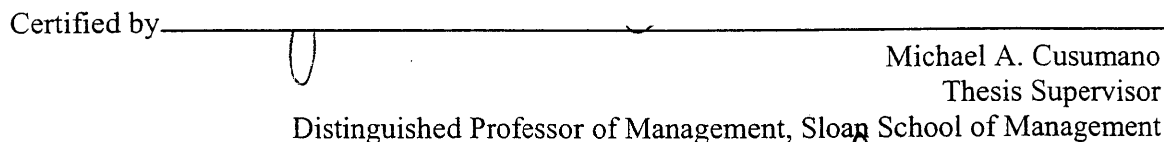

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUN 2 1 2006
LIBRARIES

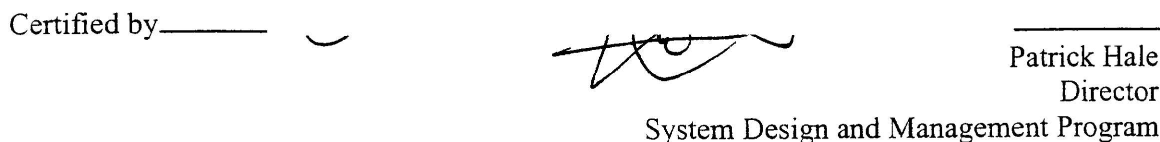
© 2006 John A. Hempe
All rights reserved

The author hereby grants to MIT permission to reproduce and to
distribute publicly paper and electronic copies of this thesis document in whole or in part.

BARKER


Signature of Author
John A. Hempe
System Design and Management Program
June 2006


Certified by
Michael A. Cusumano
Thesis Supervisor
Distinguished Professor of Management, Sloan School of Management


Certified by
Patrick Hale
Director
System Design and Management Program

Product and Program Management: Battling the Strangler Trees of System and Social Complexity in the Software Market Jungle

by

John A. Hempe

Submitted to the Engineering System Division's System Design and Management Program on June 9th, 2006 in Partial Fulfillment of the Requirements for the Degree of Master of Science in Engineering and Management

ABSTRACT

An exploration of Software Product and Program Management as recently emergent roles in the information technology sector is presented. The exploration is presented in six sections divided into two major parts. The first part, in four sections, explores Product Management from a primarily anthropological and managerial perspective, while the second part, in two sections, explores major engineering issues related to the role.

The first part gives a synopsis of the history and economics of software products, demonstrating the rapid evolution of a field facing unprecedented problems with product complexity and motivating the need for Product Management. The role of Product Manager is explored in detail using both extant literature and interviews conducted with current practitioners in industry. The related role of Program Manager is briefly discussed. Finally, an extended historical case study is presented demonstrating the struggles and pitfalls of Product Management in software product companies.

The second part explores two major engineering issues related to the Product Management role: Project Management and Requirements Engineering. A survey of major Software Project Management methods in use is given along with critiques of their effectiveness. Finally, the emerging field of Requirements Engineering is studied, with the conclusion that purely analytical methods such as semi-formal modeling cannot obviate the need for social process methods. Such methods take into account the tendency for human communication problems both to sabotage and to become embedded within software systems.

Thesis Supervisor: Michael A. Cusumano
Title: Sloan Management Review Distinguished Professor of Management

Acknowledgements

This thesis would not have been possible if had I not been accepted into the MIT community, which I have dreamed of joining ever since I watched a ping pong ball-gathering robotic competition held here on video in a high school science class at the age of 14. I am grateful for the experience.

In particular, I would like to thank my advisor, Professor Michael Cusumano, who provided guidance for this thesis. "I have many sad stories about software companies," he once said in a lecture for his Software Business class, pausing and looking off as if into the distance, or into a dark abyss. "Sometimes I wonder why I continue to study the subject." He paused a moment longer, as if searching that abyss of uncertainty for a hint of the light from software's first glory days, then continued, seeming to find it: "But I still find it fascinating. Software can change the world."

I thank Professor Daniel Jackson of CSAIL for donating his time to interview for this thesis, and for allowing me to record the interview. I also thank the industry interviewees who donated time providing invaluable insight into the current practice of Software Product Management. In alphabetical order, they are: Scott Case, Product Manager at Atlas Solutions; Philip DesAutels, former Product Manager and current Academic Liaison at Microsoft; and Shuman Ghosemajumder, Business Product Manager of Trust and Safety at Google. Thanks to Professor Nancy Leveson for her excellent "Software Engineering" course which provided a lot of relevant material.

In closing, I would like to thank some personal connections. I thank my parents Jeff and Linda, who provided hours of emotional support via telephone. Also, I thank my long suffering fiancée Jennifer, who put up with our 3,000 mile separation for a year and a half. Thanks to Dr. Stephen Harrison, who helped me get into the SDM Program and who supported me when I first came to Massachusetts, and to my California dot com bomb buddy Karl "Wes" Chester. Lastly, I would like to thank my oldest and 24 karat gold friend Gerald "Jerry" Richmond, whom I met in Kindergarten and have known all my life, for our hours of off-the-wall Internet chats during the writing of this work, which helped keep me sane.

Table of Contents

Abstract **2**

Acknowledgements **3**

Table of Contents **4**

Preface: 8 Years in Silicon Valley **6**

I. Introduction **9**

II. The Software Product Problem Arises: History, Economics, and the Strangler Tree **13**
- Factors Motivating Product Management 13
- Godlike Powers and the Strangler Tree 13
- The Emergence of Product and Program Management 18
- The Economic Landscape for Software Products in the Internet Age 22

III. Understanding the Roles of Product and Program Manager **29**
- Approaching the Target Roles 29
- Inbound versus Outbound: Listening versus Talking to the Market 29
- Industry Perspective on Product Management 31
- Deliverables of the Product Manager 37
- Demographics: Who Are Product Managers? 40
- Program Management 42
- The Program Management Grey Area 44
- Conclusions on Product and Program Management 45

IV. Struggle and Failure in Software Product Management **48**
- An Extended Case Study: Netscape Navigator versus Internet Explorer 48
- Judo Strategy 53
- Uncontested Ground 54
- Be Flexible 55
- Marketing Warfare in Product Management 62

V. Software Project Management **64**
- Waterfall Model 64
- Spiral Model 65
- Agile Software Development 67
- Capability Maturity Model for Software 75
- Process, Culture, and Project Management 79

VI. Requirements Engineering **82**
- The Product Management Frontier 82
- The Werewolf Theory of Requirements Engineering 87
- Capturing Requirements with Social Process Methods 93
- Engineering Software Requirements in the Future 102

VII. Conclusion **104**

Bibliography **108**

Appendix A: The Great Windows / UNIX Platform Battle and Software Commoditization **110**
Parallel Timelines in Hardware and Software 110
The Mainframe to UNIX to Open Source Timeline 110
The Intel to IBM / Microsoft to Wintel Timeline 112
The Non-Euclidean Software Market: Do Parallel Lines Ever Meet? 114
Ubiquitous Networked Computing 115
Unintended Consequences and Open Source 115
Timeline Collision and Fallout 116
The Resultant Software Product Black Hole: Commoditization 117

Appendix B: Industry Practitioner Interviews **118**
Scott Case, Atlas Solutions 118
Philip DesAutels, Microsoft 122
Shuman Ghosemajumder, Google 126

Appendix C: Interview with Professor Daniel Jackson **132**

Preface: 8 Years in Silicon Valley

A thesis is personal, at least in terms of motivation, and I have chosen to use this section to speak briefly but personally about my experiences leading up to the writing of this work. Only a sketch, of course, and a lot is left out, but then this is a preface and not a memoir.

Once upon a time, in 1999, every company, startup, dot com idea, dynamic duo of a programmer and an MBA, almost any organization engaged in the production of software--especially Internet related software--was viewed as a potential gold mine, or as an oil drilling operation near-certain to end up a gusher. Irrational exuberance ruled the day. I lived through the days of euphoria, and struck it rich (temporarily) in the days when Internet-based enterprise software could do no wrong. My company, BroadVision, Inc., peaked as an S&P 500 company with a $29 billion market cap. Venture capital money poured into anything Internet or "dot com" related.

Humanity does this sort of thing every so often. We all have a weakness for the potential quick buck; coincidentally, the heart of the Internet gold rush was in roughly the same place as the literal Gold Rush of 1849: in San Francisco and the surrounding Bay Area. Yet the surface gold, easily accessible, quickly ran out, and many gold seekers died paupers. According to one historical site, "James Marshall - the original discoverer of gold - died on his claim in the gold fields in 1885 without even enough money to pay for his burial." [1] Although hopefully most of the Internet gold rush participants would have more luck rebuilding their future, many found their wealth decimated or destroyed.

After what I call the Dot Com Bomb, by late 2001, businesses were going down left and right. January 8th, 2001, I walked into my office on a Monday morning at a customer relationship management (CRM) company called North Systems. I was a consultant, charging $110 an hour as my own agent, the kind of money I never dreamed one could get without being a lawyer. I had just taken a long holiday vacation. My cubical had been moved, and all my equipment was unplugged. I found my manager.

"What's going on?" I asked.

"It's over. The company's dead."

I looked into his eyes, and saw that they were slightly wild. In a second, I noticed he was unshaven, haggard, with a light sheen of sweat on his forehead. He clearly wasn't joking. He had worked 50 – 60 hours a week for this wildcat startup for two years, and would walk away with nothing. What could I say?

Amazingly, they paid my final bill. Yet it was the beginning of the end of my 8 year career in the San Francisco Bay Area. Company and product launch parties were soon displaced by "Pink Slip parties" for laid-off IT workers, and these parties often had lines out the door. Contracts suddenly became scarce and rates deflated drastically. Startups were dying even faster than they once popped up just a year before. What had gone so horribly wrong? Part of it was certainly just human nature, the Gold Rush mentality gone wild.

Yet something deeper was wrong with the business of software, something that plagued businesses before the Internet boom and continues to plague them today. Fred Brooks knew part of the story by 1986, as virtually every software manager knows: there is "No Silver Bullet" to ease the creation of software.

Software is hard.

Less pithily but more descriptively, the creation of wealth through the writing, marketing, and selling of software as a sustainable business is incredibly difficult. This is what I learned during my triumphs and struggles in the Silicon Valley, as I tried to keep my head above water programming incredibly complex systems while watching the NASDAQ slowly (then quickly) drown.

My own insight is not, "software is hard." My insight, rather, is this: In spite of the painful lessons of history, managers at all levels in business today somehow persist in believing, as evidenced by their actions, "software is easy." Managers consistently underestimate the difficulty in implementing a given set of functionality. They consistently underestimate the expense of maintaining a system. They underestimate both the importance of and the difficulty with intra-corporate communication, especially between Engineering and Marketing, about what a software system actually does, could possibly do, and should do in the future. A general distrust between Engineering and Marketing is endemic (one might say epidemic) in the software industry.

Communication channels between the two are often neglected, and a silo effect occurs, devastating to a software business dependent on a well marketed product.

These are the problems which motivated this thesis. After experiencing the lofty highs and crushing lows of the Silicon Valley software business, I've become sensitized to the fact that commercial software is far from a MMOP or a "Mere Matter of Programming." Human factors, business factors, can easily prevent the best engineered software from becoming a market success. My hope is that this thesis will shed a little light into the abyss between the engineering and the business, especially the marketing, of software.

I. Introduction

Software has exploded onto the stage of virtually every aspect of public and private life in the past half-century. With origins in publicly funded military and aerospace systems, software quickly trickled down into universities, privately held businesses, hospitals, homes, cars, stereos, microwave ovens, digitally encoded neo-Walkmans such as the iPod, cell phones, and, of course, our now-ubiquitous Personal Computers. To paraphrase MIT Professor Nancy Leveson, the advent of microprocessors capable of running general-purpose software has given us the capability to build a single generalized machine and create a near-infinite number of specific machines from it.

Corporate or enterprise software has also become critical to the running of virtually all major businesses in the modern world. The 2003 Chaos Report indicates that, unfortunately, only 34% of IT projects were considered successes, with only 52% of originally allocated requirements appearing in finished projects.[2] One paper by Christof Ebert of Alcatel claims that there is a "Bermuda Triangle" effect between the marketing, strategy, and technical aspects of software development into which product success gets lost due to the lack of agenda coordination.[3] The extremely rapid pace and inherently complex nature of software development makes total life cycle management of a product's business value—Product Management—uniquely challenging in the software / IT sector. Even a small incremental improvement in the area of software product management could yield great dividends in a field experiencing a project success rate well under 50% as virtually all sources agree.

The central theme of this thesis is software product management, a topic about which surprisingly little has been written. The role is probably the most strategic and

cross functional non-executive level (or C-level) role in the modern software corporation. Due to the highly cross functional nature of the Product Management role, it is necessary to explore the "satellite themes" of Program and Development (or Project) management. The original goal of this work was to draw a bright line around the Product Management role, exhaustively explaining all duties, the ideal toolset, and at least a skeletal outline tantamount to an instruction manual for the performance of the job. Exploration into the role slowly revealed that this task is about as tractable as writing an explicit manual entitled "How to be a Good CEO." There are few bright lines and pinpoint answers to be found, no list of command directives to be a good Product Manager.

Thus, we explore the role as if tightening a perimeter around elusive prey. First, we motivate the role from a historical perspective, examining the rapid rise of the new class of highly complex software products, and the emergence of the Product and Program Management roles in the software industry. We briefly examine the intense hype of the Internet gold rush, and the terrible crash back to reality, as well as some of the economic factors unique to software products. These factors demonstrate the difficult and fast-paced environment creating the need for the Product Manager. They do not define him exactly, but quarry, if you will, the block of marble where he exists from the surrounding rock of a difficult market. We then chisel at that block, exploring more directly the role of Product Manager in an attempt to sculpt a likeness. Lacking a tight, abstract definition of the role, we turn to modern industry. Interviews with three prominent practicing Product Managers help triangulate a definition of the role as practiced today. We'll also look at the overlapping role of the Program Manager, emerging as if fashioned by Microsoft from a rib of the Product Manager. (No gender

bias is implied—if anything the Program Management role is slightly skewed male compared to the Product Manager role, which splits about two-thirds male to one-third female).

After directly exploring Product and Program Management, we examine in detail a historical case study, attempting to illuminate the roles further in a recent historical "Clash of the Titans" in software products. The David and Goliath browser wars of Netscape versus Microsoft demonstrate the strategic maneuvering of two powerful foes in attempting to position their products and drive them into market dominance. These are precisely the broad strategic issues Product Managers face in their daily lives, although often on a somewhat smaller and less public scale. Yet Product Managers have to make smaller scale versions of strategic maneuvers exactly like those of Gates, Barksdale, and Andreessen, because in many companies the Product Manager is truly the CEO of the product.

The final third of this work focuses on engineering methods relevant to Product and Program management. Both roles are involved with the discipline of Development or Project management. First we examine a topic at the cutting edge of practice, the controversial and intensely studied emergent discipline of Requirements Engineering. We take a brief look at the current practice of RE today, and then examine several emergent methods, including a detailed examination of a semi-formal method under research in the Computer Science and Artificial Intelligence Lab at MIT. Finally, in consideration of the Product Manager's duty to schedule or at least monitor projects, we delve into a survey and review of both time-tested and recently emergent Project

Management methods, examining how they affect company culture and the role of the Product Manager.

We conclude with a retrospective on Product Management: what we do know, what is still uncertain, and what aspects of the role will likely never attain the status of a "solved problem." In the final analysis, we can learn a great deal from industry practice and continue to improve tools and methods, but no replacement of the need for vision and leadership in the role (and in the related roles of Program and Project management) is foreseen.

II. The Software Product Problem Arises: History, Economics, and the Strangler Tree

Factors Motivating Product Management

As outlined in the Introduction, it is difficult to draw a "bright line" around the Product Management role. This section explores historical and environmental factors motivating the creation of Product Management as a role in the firm. First, we delve into a philosophical exploration of the awesomely complex nature of software systems, which translates into complex software products. This is followed by a selective elucidation of the frenetically paced history of software products, with winner-takes-all battles requiring superior management for mere survival, let alone success. The timeline and historical emergence of Software Product Management and invention of Program Management are briefly studied.

We conclude with a pass over a detailed slice of software history, with an eye to the associated economic forces, that being the recent Internet boom and bust. These economic factors do not directly define the Product Management role, but are critical to understand as they define the strategic environment in which modern Product, Program, and Project management take place.

Godlike Powers and the Strangler Tree

Mankind has not yet wrapped its collective mind around the possibilities software creates, which makes sense as software is possibly the most powerful single technology invented for the extension of that mind's power. Software is pure design embodied; by far the closest man has come to achieving the godlike power of thought made flesh. In

existence for less than a century, software has revolutionized mankind's pursuit of scientific knowledge (if not wisdom), enabling him to, for instance, land a remote controlled robot on Mars and scan, at least to a first, gross level, his own genome.

Of course, godlike powers do not generally come easily or without peril. Intercontinental ballistic missiles capable of carrying nuclear warheads accurately across the globe to annihilate most of humanity still sit quietly in their silos, ready to fly and find their targets using software. Hellfire missiles launched from pilotless airborne vehicles controlled remotely through a television screen, and many similar devices, could not exist without software and make even conventional war and killing perhaps far too easy and impersonal. Creation of deadly chemicals, biological weapons, and a host of other evils can be made possible or at least radically accelerated by software. All technologies of great power can be wielded to the great good or terrible ill of humanity, and somewhere along the line generally end up getting used for both.

Yet perhaps the greatest difficulty mankind faces in the future of software is not the question of how it is used so much as how it evolves. Sheer complexity is the enemy of software. Complexity creeps into large software systems like a slowly spreading disease. Like the strangler trees of the equatorial forests, complexity softly slithers its way into the subroutines and interfaces of systems, seeming innocuous until one day a major modification must be made to the system. Engineers merrily continue to develop the "feature of the day" until that moment, like new branches and leaves growing on the tree. Eventually, however, the day comes that the unavoidable modification must be made: a major change to the intent and functionality of the system.

In nature, the strangler tree (see Figure 1) eventually kills the underlying host tree, choking it to death while the strangler lives on, holding the same shape as the original tree. Similarly, complexity eventually kills all software systems of sufficient size: there is still life, the system is still somewhat functional, but it simply becomes impossible to alter the intent or basic functionality of the system to any significant degree. The power of software may be potentially godlike, but we human beings do not have the minds of gods.

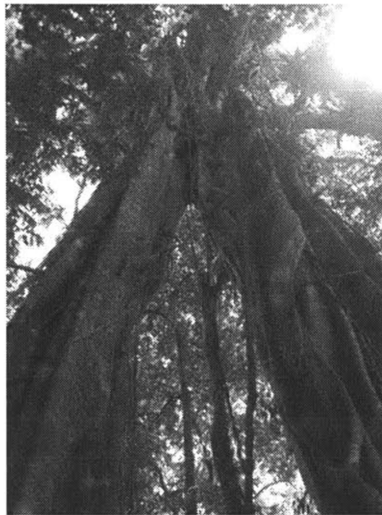


Figure 1: Strangler Tree in Mosman Gorge, Australia (S. MacGregor, 2002) (Source: http://www.pbase.com/mscottnscp/image/10019085/large)

In fact, we humans have rather strict cognitive limits. The reader may be familiar with the now-near-proverbial "7 plus or minus 2" rule stating that most people can hold five to nine items in short term memory. Software systems these days can easily run into millions of lines of code with thousands or tens of thousands of subroutines, methods, function calls, interfaces, and other intangible entities to keep track of. The United States Air Force's Theater Battle Management Core System is one example of a system having major issues with the software strangler tree. It is a mega system trying to integrate a

huge diversity of "strangler tree" systems: mature military systems rife with unmanaged complexity whose intents and functionality can no longer be significantly changed.

What forces exist to counter such complexity? To counter the problem of strangler-tree systems? In a word: competition. In two words: Commercial competition. Competition provides the motivation for mankind to produce software systems that overcome the strangler tree problem. This may seem a naïve or even base idea at first. Competition? Surely there is a more scientific answer than that! Some sort of information theory, complexity theory, set of mathematical formulas, the calculus of software, something?

Perhaps there would be if software was a "mere matter of discovery", but software is a combination of discovery and invention. The key, then, is not a particular tool to solve the problem, but a robust and continuous *motivation* to keep pursuing better and better solutions. That motivation is commercial competition. Nothing motivates men like the creation of wealth. Pure curiosity certainly plays an important, and perhaps some might consider a nobler role, but ultimately a man has to pay his bills. Bill Gates was perhaps the first, or possibly just the most successful, person on the planet to realize that software is fundamentally as commercial as any other product: that people and businesses should and would pay for software. It is notable that, for all the references to Microsoft as "the evil empire" and complaints about its monopoly power, it is the software of this company—which approached software as a *business*—that truly first drove software into the common consciousness of the entire first world and continues to do so today, now penetrating developing countries.

For software to work as a *business*, one thing has to exist: willing buyers. If software does not sell, the expensive work of creating it will eventually put the originating company out of, well, business. Software must create value for a buyer. Whether that value is entertainment value or increased efficiency of running a business or analysis of equities or pork bellies to facilitate better buying decisions, there must be some perceived value in the software or it will never be purchased. Just as important, the customer must actually realize value after the purchase lest the first purchase become the last. In any other business, such statements would be considered tautologically obvious, but software has an extremely strong culture of engineering over marketing, of programmers over businessmen.

There is a pervasive and dangerous attitude in the industry: "If we build it, they will come." The "they" in question are customers, *paying* customers. Unfortunately, the world of paying customers is not like the Field of Dreams, and history has demonstrated—rather harshly at the turn of the millennium—that "they", the paying customers, most certainly will not come to whatever a business haphazardly decides to build. Nonpaying customers may come in droves if something is mildly interesting or strikes their momentary fancy, giving rise to such fluffy terms as "eyeballs" and "mindshare" in the heady dot com days, but these people melt away like the morning dew once the idea of paying comes into the picture. Those dot com operators would have done well to realize that "nonpaying customers" is really an oxymoronic phrase. Anyone who does not pay for a product is not actually a customer at all.

Thus, we arrive at a major point this thesis addresses. The software business needs a greater respect for the discipline connecting written software with paying

customers: the discipline of marketing. The business must abandon the old ways of building supposedly cool stuff and hoping someone will buy it, and must instead actually listen to what the market wants. This is not always easy, as software is tricky and figuring out what the market wants may turn into a delicate process of both educating and then listening to the potential customers. Yet performing this delicate process correctly can make or break a software business.

Now that we have set the philosophical stage, we will turn to history for an examination of the emergence of the key Software Product Management role. (We will see how the derived role of Program Management emerged in the next chapter.)

The Emergence of Product and Program Management

Traditional Product Management

The job and role of Product Manager is an artifact of modern business. There were no Product Managers in any industry in the nineteenth century or before. Of course, a business historian could come up with a wide array of examples showing that somebody was doing Product Management for firms earlier in history, but the role of Product Manager as a full time job simply did not exist yet. The first company that came to mind when researching pre-computer Product Management was Proctor & Gamble. William Proctor was a soap maker and James Gamble was a candle maker. They joined forces in 1837, and by the time of the American Civil War, landed a deal to provide the Union Army with soap and candles. Today, P&G is a large consumer goods concern with many, many famous brands from Pampers to Pringles, Crest toothpaste (Figure 2) to Charmin toilet paper.

Figure 2 Crest Whitening Expressions (Source: http://www.crest.com)

Most sources attribute the P&G with the pioneering of Product Manager as an independent role worthy of a full time job in business. One Internet career source notes:

> The use of product management dates back to the 1950's. Proctor & Gamble was one of the first companies to use and develop product management positions. Initially, Proctor & Gamble had several products within their own company that were competing against each other. Today, companies generally use their products to work together for the market share and compete with products from other companies. [4]

Along Came Technology

Product Management made a lot of sense for Proctor & Gamble with its straightforward, physical, mostly consumable products. The customer will wash his hands or body with the bar soap, do the laundry with Tide, brush his teeth with Crest toothpaste, and shave his face with the Gillette razor and its complement, Gillette Foamy shaving cream. Selling these products to a maximum number of customers for maximum profitability is certainly not a simple problem, but there is, at least, a fundamental underlying product with essentially static functionality. Crest toothpaste, for instance, has reached a level of extremely fine market optimization, but even the Crest Whitening Expressions Mint toothpaste is still a tube of toothpaste used to brush the teeth. Shuman Ghosemajumder, a Product Manager at Google, noted in his interview (Appendix B), "In high tech products, there aren't a lot of features as in toothpaste. For instance, customers

might like baking soda or other whitening agents, but then it's really about market segmentation."

High technology products, especially software, have less clear utility. Software is innately intangible, often difficult to explain, and is perceived as extremely malleable in functionality. These factors combine to make software highly susceptible to positioning manipulation. Unfortunately, especially when it comes to malleability, these factors also combine to make it difficult to position software at all, or to position software such that customers and the software pass each other like two ships passing in the night.[5] Compared with traditional, P&G style products, software product management requires an extremely high level of finesse and an increased focus on inbound versus outbound marketing issues. More on those issues in the next chapter, when we study the role and industry perspective in detail.

Product and Program Management Emerge at Microsoft

Everyone knows that Microsoft is ascendant in the world of prepackaged software products, and that the Windows operating system is dominant on Intel-based PCs. The short story is that this happened because Microsoft took software seriously as a product, and more seriously than hardware, early on. They were one of the first serious users of Product Managers and they invented Program Managers. The Windows / Intel timeline shown in Figure 3 shows the emergence of Product and Program Management at Microsoft long before they achieved true desktop dominance with their "hit" Windows 3 operating line. The Program Manager role, as invented by Microsoft, was a technical or engineering / development-bound version of the Product Management role and will be examined more closely in the next chapter.

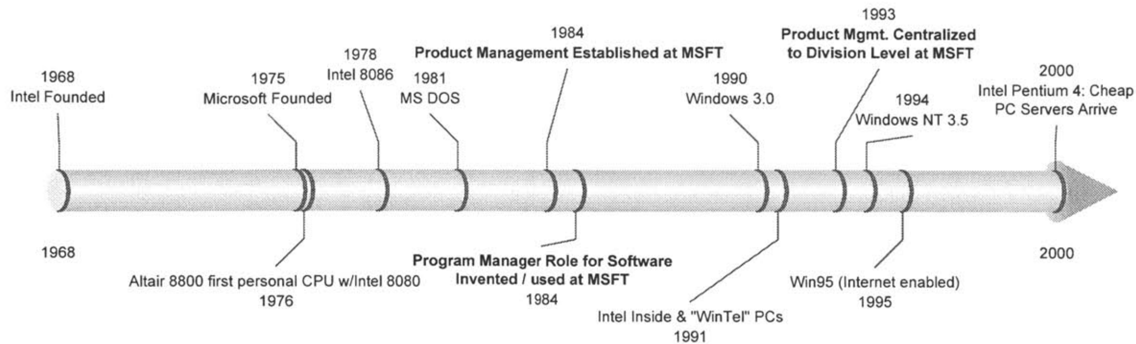


Figure 3 Product & Program Management Emerge at Microsoft

There is a longer story here, of another and parallel timeline involving UNIX and Sun Microsystems. That timeline seemed to be losing relevance for awhile until the advent of Linux and the new vitality of Open Source. Not everyone, especially younger IT professionals, understands how this came to be. Why is this relevant to software Product Management? Both the markets for software products, and the stage of the competitive landscape, have been set by these powerful, parallel trends in software & hardware history. The future stage and market, furthermore, will be to some degree determined by the trajectory of these continuing timelines. A detailed study of these parallel timelines in hardware and software is instructive for every Product or Program Manager, but is peripheral to studying the history, practice, and future of the roles themselves. Deeper looks at the timelines, their collision, and its implications for the software market, including Open Source and commoditization, especially enterprise software commoditization, are given in Appendix A. What we *will* study in more detail in the "main line" here is the recent and very difficult landscape set up for Product and Program Managers by the hype and crash of the Internet Boom, and some important economic factors unique to software products and, therefore, to software product strategy.

The Economic Landscape for Software Products in the Internet Age

Combinatorial Innovation and Financial Speculation: the Dot Com Boom

As in any Gold Rush phenomenon, whether the literal rush of 1849 or the figurative Dot Com Boom of the late 1990's, a group of people or sometimes an entire society indulges in the illusion that there is some unlimited new resource capable of generating wealth for virtually zero or nominal effort. For a brief time, the philosophy seemed to be, simply program anything Internet or Web enabled and make tons of money. A prominent Silicon Valley venture capitalist described the dramatic run-up in technology stocks as the greatest legal creation of wealth in human history. However, not all of it was legal and not all of it was wealth. Virtually everyone involved in high tech, or stock market investment of any kind, has seen some version of the chart in Figure 4 showing the NASDAQ bubble and crash:

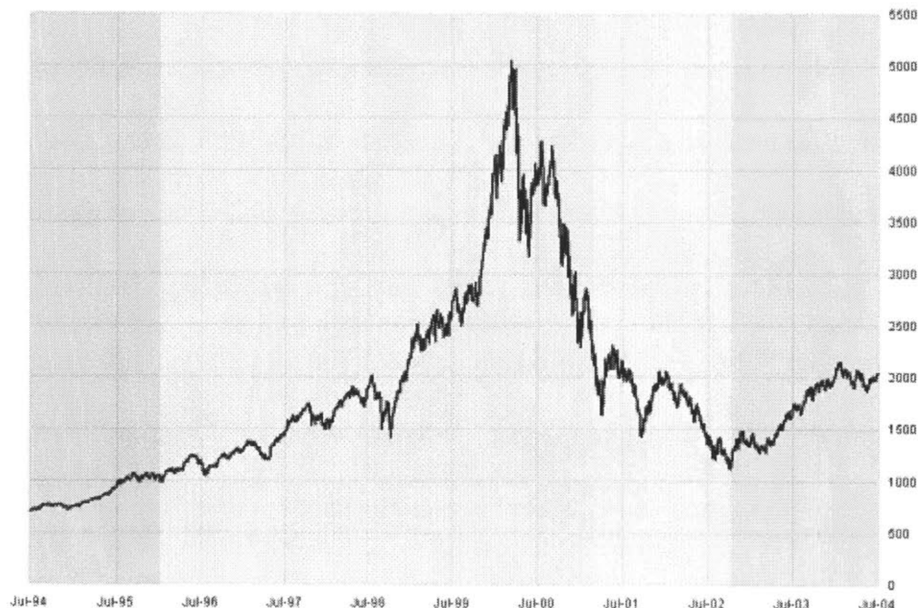


Figure 4 Internet Stock Bubble & Crash (Source: http://www.ehsco.com/misc/economy/nsdq-decade.gif)

The NASDAQ Composite lost 78% of its value as it fell from 5046.86 to 1114.11 (www.investopedia.com). How did software products drive this bubble to appear? Economist and information technology specialist Hal R. Varian, a professor at the Haas School of Business, believes it is an instance of what he calls *combinatorial innovation.*

> Every now and then a technology, or set of technologies, emerges whose rich set of components can be combined and recombined to make new products. The arrival of these components then sets off a technology boom as innovators work through the possibilities...The attempts to develop interchangeable parts during the early nineteenth century is a good example of a technology revolution driven by combinatorial innovation. [6]

Such combinatorial innovation occurs in waves or clusters, including the wave of weapons manufacture in New England which led into a wave of domestic appliance innovation in the early 19th century. Professor Ed Crawley at MIT has pointed out that the entire field of Mechanical Engineering arose from the wave of innovation surrounding the invention of the steam engine, which spawned the Industrial Revolution and eventually became a dominant technology in the shipping industry. The internal combustion (gasoline) engine represents another such wave.

Each of these waves is accompanied by a speculative investment boom. The difference, for the Internet revolution, lies in its magnitude and compressed nature. Though the intense stock market spike was completely out of proportion to business reality, the fact is that an amazing amount of real innovation was going on at an extremely rapid pace. Varian believes this phenomenon occurred because, "the component parts of the Internet revolution were quite different from the mechanical or electrical devices that drove previous periods of combinatorial growth...they were 'just

bits.' They were ideas, standards, specifications, protocols, programming languages, and software." He notes, "Unlike gears and pulleys, you can never run out of HTML!" [6]

The combinatorial innovation of the Internet boom was driven by a fluidity of ideas that has never been seen in the realm of physical product creation. For instance, open source software is like what Varian calls, "primordial soup for combinatorial innovation. All the components are floating around in the broth, bumping up against each other and creating new molecular structures, which themselves become components for future development." Before the Internet, one cannot imagine similar phenomena: Open shipbuilding? Open lightbulb design?

Real value was being created in the wave of Internet software related combinatorial innovation, but speculative investment can always outstrip the creation of real value as the market responds to the forces of fear and greed: in this case, greed ruled for a short but intense time. Investors tried to rationalize the mad pace of investment and price multiples being paid for fledgling companies by claiming that a New Economy had emerged. Though it is doubtful that many actual economists believed in the New Economy, investors and technologists alike clung to the idea in droves.

The "Old" New Economy for IT

On the topic of the bubble's magnitude, Varian says little, simply noting that a lot of "dumb money" comes into the stock market when the public gets excited about a new technology. He also does not believe in the New Economy, a notion most investors probably became disillusioned with as they were brought down with the crashing NASDAQ. His notion on this topic is that the so-called old economic principles work, but some become more emphasized in the information economy: "Effects that were

uncommon in the industrial economy—network effects, switching costs, and the like—are the norm in the information economy." Thus the economy is not new *per se*, but certain principles do apply differently:

- Second-order effects for industrial goods are often first-order effects for information goods.
- High fixed costs and virtually zero marginal costs are a hallmark of the software product industry. (Varian notes that this is an idealization for most physical production processes, but is the baseline case for software products.)
- First-degree price discrimination: IT allows, in the extreme case, for a "market of one" or mass customization / personalization
- Switching costs are generally high in the realm of software. Switching from Windows to Linux, for instance, can be very costly.
- Competition for new customers is intense due to the high level of customer lock-in.
- Cost of search for consumers can be dramatically lowered on the Internet.

As of this writing, with the company Google having recently gone public, search is the number one force driving the post-crash Internet business. Search in this context goes beyond economic search, though it is notable that the search-based business model thus far depends entirely on advertising—essentially a form of providing customers search information via a push model. For retailers, attempting to achieve opaque pricing becomes challenging in the Web world: Ellison & Ellison have found that online retailers often engage in "bait and switch" tactics online: advertising an inferior product at a low price to attract users to their site, attempting to confound such new technologies as shopbots and agents that compare prices from multiple vendors. [7]

Certainly we have not examined all of the counterintuitive economic effects in the software industry, but this survey of some of the major effects gives an indication of the

tricky strategies related to economics which Product Managers face. We will examine one more economic aspect of software products before moving on, that being product interdependency for economic viability.

Economics of Systems Effects in Software

Software products have a high degree of *systems effects*: products are often useless unless they are combined into a system with other products. This phenomenon is so prevalent in software that many companies targeting the business to business market sell software "systems" rather than "products". Even shrink-wrapped software products, the most self-contained software available, have many systems effects, especially now that network capabilities are expected nowadays. Take a retail PC video game: the consumer needs a PC to play the game, a CD-ROM to read it, the Windows OS installed, most likely a 3D accelerator graphics card, and for an increasing number of games, Internet connectivity. TurboTax from Intuit, once a completely self-contained program subject only to the most basic hardware system effects, now requires Internet connectivity to download a State return as well as to file returns (if low-hassle online filing is desired).

Software as part of systems, Varian points out, causes companies to *Integrate, Collaborate, Negotiate, Nurture,* and *Commoditize.* [6]

- **Integrate**: One complementor acquires the other, a common Microsoft practice to the point of raising Department of Justice resistance.
- **Collaborate**: Revenue sharing, such as the famous case of Blockbuster's software-enabled revenue sharing between studios and video stores (rentals were provided to stores for $0 to $8, with rental revenue split roughly in half with the studios, allowing Blockbuster to stock many more copies and reduce rental stock

outs). This arrangement would have run into unenforceable contract problems if not for computerized record keeping.

- **Negotiate**: One firm commits to cutting its price if the other firm also cuts its price.
- **Nurture**: One firm works with the others to reduce their costs. Adobe, for instance, works with printer manufacturers to ensure that they can effectively use its technology.
- **Commoditize**: One firm attempts to stimulate competition in the other's market, thereby pushing down prices. Microsoft has established the Windows Compatibility Lab to ensure that hardware manufacturers all produce to a common standard. This helps facilitate competition, pushing down the price of hardware. (Microsoft wants cheap hardware on which to run their complement, software.)

All of these forces derived from systems effects help reduce prices, partially counteracting the tendency in the software industry toward "high industry concentration ratios and monopoly power." Complementors, according to Varian, "may sometimes play a similar role" to competitors in monopolistic situations for price discipline. [6] Still, a high degree of industry concentration exists, most notably Microsoft's domination of the PC operating system and applications environment, but also visible in other areas, such as Oracle's domination of the RDBMS market. The trend of a few unique and highly consolidated winners, however, is not new to the software industry. A similar phenomenon occurred in auto manufacture, for example, as hundreds of competing manufacturers concentrated into the few large players today, now competing mostly against foreign imports.

Completing the Historical and Economic Picture

Subject to extremes of hype, overinvestment, standards battles, and systems effects, we can see that the world of software products is fraught with peril. The need for expert strategy to navigate such a difficult, highly competitive environment in which forecasts are almost universally uncertain should now be obvious. Though no one can keep all factors in mind all the time for every business decision, the management professional with a good concept of the software industry's historical and economic *gestalt* has great advantages in navigating the unstable landscape. With the historical and economic environment clearly, though far from exhaustively established, we are ready to move on to a direct examination of Product Management and the derived role of Program Management.

III. Understanding the Roles of Product and Program Manager

Approaching the Target Roles

With our discussion of the unique historical and economic environment surrounding software products complete, it is time to examine directly the role of the Product Manager and its derived cousin, the Program Manager. We have seen how the role emerged, migrating and morphing across industries. Now, we'll first take a look at the unique new need for a focus on inbound versus outbound marketing as a job responsibility in this role as redefined for high technology. Second, we delve into industry perspectives researched for this work via interviews to further explore and define the role as practiced in the new millennium. Third, we look at some concrete responsibilities of the Product Manager, especially with respect to documentation deliverables, as defined in the literature. Fourth and lastly for Product Managers, we take a quick statistical overview of who Product Managers are from a demographic perspective in modern technology companies. Finally, we look at the more technical, Development-bound role of Program Management. Actual Project (or Development) Management will be discussed later, in its own section, focused more on methodology than the job and role itself.

Inbound versus Outbound: Listening versus Talking to the Market

A primary distinction between the activities of a Product Manager became apparent early in research for this work. The distinction is between marketing activities considered "inbound" and those considered "outbound" by marketing professionals.

Inbound issues involve listening to the market and designing product feature sets that can deliver value based on what the market is telling a company. This does not necessarily equate to what end users say they want, but to what market data actually supports. Many customers, for instance, potential end users in a corporation, are "mouth hungry" for features, but this does not necessarily mean the decision makers in their company will support actual purchases based on those features. Interacting with engineering is generally considered part of the inbound side of the product manager's job, and may be quite dominant for some product managers, especially at more mature organizations.

The outbound side of the PM's job is the more pure marketing-oriented function, involving talking to the market in various ways. One book on the subject defines the outbound / inbound split this way:

> *Product Marketing*: Writing collateral and white papers, dealing with sales issues, providing support, talking to analysts (often called outbound product management).
>
> *Product Management*: Defining product requirements, interfacing with engineering (often called inbound product management).
>
> One person does both tasks in many startups...The combination provides a manager with broad perspective; additionally, having one person lead both areas is efficient and reduces the chances of misunderstanding or miscommunication of the priorities required to shepherd the product. [8]

One set of related companies researched for this thesis, the aQuantive group, including Atlas Solutions and Avenue A, had very specific division of labor. This included a specific Product Management role, for inbound marketing, and a Product Marketing management role for outbound responsibilities. Of course, these companies are specifically Internet marketing companies and thus very focused on the marketing

aspect of business. One would expect to find the highest level of role differentiation in marketing for a business that has this as its core competency.

Industry Perspective on Product Management

Ambiguity in the Software Product Management Role

At Microsoft, a stagnant company in terms of stock growth recently (relative to 2006) but still essentially the Proctor & Gamble of software products, the inbound / outbound division is not necessarily as decisive. Bill Shelton, a Product Manager at Microsoft who submitted a brief "email interview" for this case, had the following to say about his role at Microsoft (Appendix B):

> My job involves the following activities: market opportunity analysis, revenue forecasting, product planning, technology roadmapping and outbound marketing messaging and positioning definition and execution. Product Management is a very broad job title at MS that spans everything from very internally focused business management activities to exclusively focused marketing and communications activities. It varies by team and specific position.

Philip DesAutels, former Product Manager at Microsoft, concurred, saying, "Bill Shelton's is a superb definition of Product Management. He also notes of his own former role, "I was talking to the press and being quoted in six or seven articles a week." That much press contact seems to indicate a largely outbound focus. Yet he also claimed to be "making sure [a series of technologies] were in sync with industry and customer's demands," an inbound responsibility. So Microsoft, at first glance, might seem relatively "sloppy" about dividing inbound and outbound responsibilities with regard to the

marketing functions of Product Management. Cusumano and Selby draw a clearer distinction in *Microsoft Secrets*:

> *Product Managers* are marketing specialists; some work in product planning as part of the product units, although since late 1993 Microsoft has centralized most of them in division marketing groups. [9]

This distinction is consistent with Philip DesAutels' role as a cross-product champion of Web Services. This role makes sense to perform from a central marketing group. Still, neither DesAutels nor Shelton gave an impression of a tightly bounded role. The best apparent explanation is that Microsoft seems to have decided to create various strategically "sliced" Product Manager roles: some PMs bound to individual product groups, but most in divisional marketing groups. Of the latter category, some have been cross functionally bound in two dimensions, those being teams and technologies. All PMs deal, to some degree, cross functionally across teams, but DesAutels' former role indicates that at some PMs are now being bound cross functionally across technologies, rather than singly bound to, say, Excel or Office.

Even in a company with more evolved Product Management strategy than probably any other company in the world, the role definition still came across as somewhat "slushy", or at least self-determined, for both Microsoft interviewees. Literature on the topic often points toward some ambiguity in the role. One specialty book, *Software Product Management Essentials,* asserts quite a large level of ambiguity in the role:

> Many professional job titles are vague, but few are as variable as that of the "Product Manager". Even within the software industry, the definition and role of the Product Manager varies widely. In some companies Product Managers are responsible for managing the brand of the product

> or the entire marketing mix including lead generation and sales support...Some companies view Product Managers as the liaison between Sales and Engineering...In yet other companies the Product Manager is the business manager for a product or a product group. As a result, it is difficult to pinpoint a definitive role across the board for the Product Manager. [10]

In the following section, we will take a look at the difficulty in pinpointing a definitive role, and attempt to "triangulate" the definition of the role through the lens of three Product Manager interviewees from three very different companies.

The Blind Men and the Elephant

The ancient Indian fable and poem by John Godfrey Saxe, *The Blind Men and the Elephant,* will be familiar to most readers and has become a popular way to describe any phenomenon understood differently by different observers. In the fable, each blind man feels a different part of the elephant and describes the entire elephant in those terms: the one who feels its side says the elephant is like a wall, the one who feels its tusk says the elephant is like a spear, the one who feels its trunk says the elephant is like a snake, and so on. Philip DesAutels, a former Product Manager at Microsoft and current Academic Liaison, mentioned this fable about Project Management, saying, "Everyone tells you it's something different."

Product Management is like this in that a universal, objective definition does not exist. Front line jobs in a corporation that are not highly cross-functional and do not involve corporate strategy usually have crystal clear responsibilities. The jobs of contract janitor, receptionist, and telephone customer service representative are well defined. The job of software engineer has a lot more latitude but is still a fairly well understood,

mostly tactical role. Product Management, on the other hand, is closer to a C-level role in having a significant element of corporate strategy involved.

If one cannot directly observe the elephant, it is necessary to speak with several different "blind men" to try to come up with an aggregate picture of what an elephant might be like. The PM's interviewed for this thesis were all asked, "*What do you think is the most important Product Management challenge in the particular case of software products?*" The answers varied significantly (Appendix B):

> **Scott Case, Atlas Solutions:** People want 300 features and you can only do 10. There are difficult tradeoffs in implementing new features. "Who are you going to piss off who are you going to serve?" Tradeoffs involve "keeping the lights on" versus innovation. We serve 110,000 ads per second. 6 billion ads a day.
>
> **Philip DesAutels, Microsoft:** Alignment. Alignment of people, of stakeholders, business stakeholders, strategic stakeholders. Different parts of the companies depending on you or pushing you. Aligning those people together, getting those people to come together. Even if a group is outside of the company, you have to get them to align with your interests. It's difficult to distribute the job and has to come onto one person.
>
> **Shuman Ghosemajumder, Google**: Usability. There is so much complexity associated with computer applications. Software is like traditional engineering but zero marginal distribution costs. In software you can create a "million mile long bridge." (Possible, but impractical and useless.) Creating something that is actually usable becomes very tricky. IPod, for instance, has 80% market share with a small, elegant feature set as opposed to the everything-but-the-kitchen-sink approach. It's about organization, not just limiting a feature set. There will be a market for very advanced tools, but make the day to day tasks as fast as possible and allow novices to pick it up right away.

The "elephant" of Product Management's most important mission thus felt like three completely different things to our interviewees: feature tradeoffs, stakeholder alignment, and usability. The commonality between these aspects of Product Management is the high degree of strategic, cross functional execution required to accomplish the stated mission. Stakeholder alignment speaks directly to cross functional team coordination, and the other two stated missions require alignment. Feature tradeoffs require alignment between development, marketing, and the customer.

The same is true of usability. Ghosemajumder said of this cross functional challenge (Appendix B),

> As a PM, I try to bring the perspective of the common user back to the teams. I draw comparisons to analogous products, conduct usability testing—bring the product in front of actual users....Let's create features, technology, and then worry about usability? That doesn't work. The biggest trap technical teams fall into is creating products that are internally consistent and make sense to technical people, more and more specific, worst case, that make sense only to people who work at a specific company. Or products that make sense to technical people but not to non-technical people. This affects everything.

Perhaps our "elephant" metaphorically is like an intelligent connecting cell in a central nervous system, routing and transmitting signals between stakeholders, between departments, from the external market into development, and from marketing back into the external world. With the role as a strategic router and conduit established, let us look closer at the nature of this strategic role, as well as at some of the concrete responsibilities and deliverables.

Core Responsibilities

Bringing Software Product Management Into Focus

Product Management is a high level, strategic role. Many sources refer to the role of Product Manager as the "CEO of the Product." Mark Chapman in *The Product Marketing Handbook for Software* defines the role of Software Product Management as follows:

> The product manager's primary role is to serve as the "voice of the customer." In this role, a product manager is responsible for positioning, pricing, and promoting the product, as well as managing the market adoption and product life cycle. As such, the product manager "owns" the product and is ultimately responsible for its release into the market and long-term success. [11]

This definition seems to jibe with the industry interviews conducted. Philip DesAutels, academic liaison for Microsoft, said this about his former PM role with the company (Appendix B):

> My role involved product that shipped internally. I was in charge of Web Services technologies which would go into a variety of product lines. I had to manage a series of these technologies, making sure they were in sync with industry and customer's demands. I had to make sure the developers were developing what the customers were asking for. Web Services was an interoperability platform. Really a very technical marketing role.

Scott Case of Atlas Solutions said this about his Product Management role there:

> I own what happens to software in one of our four major product areas. It's called the Atlas Media Console. This is a tool that allows traffickers and planners at advertising agencies to set up, deliver, and track performance of online advertising...as Product Manager, I decide what features should be added to the product.... Here are customer needs,

> and I see that development can do maybe 10 of 300 desired features. I go out and meet with the top 25 clients, I have met with 15-20 of the top 25 clients.

The Microsoft PM is dealing with internal customers and the Atlas PM is dealing with external customers, but there is certainly a common thread in terms of the "voice of the customer" role defined by Chapman. Chapman further delineates the roles of the Product Manager as executing four main tasks:

- Developing the marketing requirements document (MRD) and related plans and papers that document and validate your marketing and sales efforts.
- Managing the product feature list.
- Coordinating the activities of the different functional groups involved in creating a new or updated software product.
- Participating in and/or running the launch and post-launch marketing activities for a product. [11]

Deliverables of the Product Manager

Strategy

As we found in the section on the Blind Men and the Elephant, the main deliverables of the PM are not physical, but strategic: making good tradeoffs, aligning stakeholders, and promoting software usability to make it more consumable by the market. As previously noted, the PM is like the CEO of the product. Other strategic deliverables include management of product requirements (discussed thoroughly later in the section on Requirements Engineering), schedule, release process, and beta management. Some might claim the release process is an engineering task, but "product management ultimately needs to work closely with engineering on this, and in some cases, drives the meetings to discuss the bugs….Eventually you are like a field doctor

trying to perform triage." [12, p. 157] Only the PM fully understands which software issues are critical to fix for a release.

The Marketing Requirements Document (MRD)

Certain physical documents help to achieve strategic goals, and can be considered more concrete deliverables. These documents are by no means standardized across companies. The main document sources agree on as a deliverable for the PM is the *Marketing Requirements Document*, almost universally referred to as the MRD. This document is like a strategic blueprint for the entire product: background, positioning, target customer, competition, potential customers, high level functional requirements, and key product value propositions. The MRD may also include information about supported hardware and operating systems, cost constraints, sales and revenue projections, and a high-level outline of product marketing activities. [12, p. 153]

MRD-Related Documents

Some companies include details of a planned marketing campaign, where others document this separately. Documents related to the MRD include product contracts (defined by Chapman as "[agreements] between marketing and development to reach a series of product goals"), product roadmaps (discussed later in the *Requirements Engineering* section), product requirements documents (PRD, specifying the product and user experience in detail) and marketing message plans. Chapman also notes some companies "don't believe in the MRD concept or believe the MRD should be nothing more than a brief outline, though in our experience companies are eventually forced to implement an MRD system out of sheer necessity." [11, p. 570]

The Functional Requirements Document (FRD)

Usually, development writes the *Functional Requirements Document* or FRD. The MRD describes the "what" and the FRD describes the "how" of a product. This document specifies core functionality and may get into specific technologies planned to implement the product. At some companies, this will be the lowest level documentation produced before engineers actually begin writing code. At others, the lowest level will be the *Functional Specification* describing technologies in detail and getting into some architectural detail of implementation such as class diagrams. In this case, both the MRD and FRD are considered "what" documents, the former from the marketing and the latter from the engineering perspective. The Functional Specification then describes the engineering "how" in detail.

Very technical PMs may be involved with the Functional Specification. However, due to lack of term standardization, some companies that either do not produce a Functional Specification or that combine the two documents refer to the FRD as the Functional Specification. The PM's job is to make sure that the FRD (and / or Functional Specification) correctly maps the requirements document in the MRD into software functionality. [11, p. 570]

Product Management and Documentation

Software Product Management is about owning the strategy for a product, not about producing documentation. The document set that should be produced varies depending on the product, and even more so, on the size and culture of the organization producing it. Various and sundry documents the PM may either produce or manage include (but are not limited to) the product development schedule, legal documents for

beta release, software documentation for beta and release, release notes, marketing communications collateral, requests for proposals (RFPs), and feature wish lists. The PM should gauge the amount of time and energy put into each document, and into documentation overall, based on its potential impact into driving the product's market success. Expending incredible amounts of energy on a meticulously maintained documentation system is useless if the PM becomes too internally focused and loses track of customer and market needs.

Demographics: Who Are Product Managers?

Before moving on to our exploration of Program Management, let us close this section with a brief examination of the Product Manager's demographic profile both as a person and in his corporate role. This section is drawn from a survey by Pragmatic Marketing, which bills itself as the "Standard in Technology Product Management and Marketing Education." [13]

According to the survey:

- The average Product Manager is 36 years old
- 87% claim to be "somewhat" or "very" technical
- 33% are female, 67% are male
- 90% have completed college, 46% have completed a Master's program

In terms of reporting structure, the typical PM reports to a director in the product management department, although many companies do not have a separate department for this function. Reporting structure varies. The following percentages do not add up to 100% because more than one reporting relationship can be true of a single PM (for instance, a PM can be in Marketing and report to a VP):

- 46% report to a director

- 28% to a VP
- 5% report directly to the CEO
- 21% are in the Product Management department
- 15% are in the Marketing department
- 12% are in Development or Engineering
- 5% are in a sales department

Many PMs have a sales support role as part of their job, with 51% training sales people and 44% going on sales calls. Only 16% report working with the press and analysts, while 79% report monitoring development projects. 77% of PMs write requirements. Technical demands of the role have increased since the dot com crash. 52% of PMs report writing detailed specifications, up from 29% in 2001. Average ratios of Product Managers hired to products and other relevant employee headcount within the firm are given in Figure 5.

Possibly the most interesting indication from these ratios is the one-to-one ratio between Product Managers and Sales Engineers, quite telling from a corporate hiring strategy perspective. Technical and inbound marketing responsibilities may be on the increase, but Product Management comes from the Sales and Marketing side of the corporation. 44% of Product Managers go out on sales calls. This raises a problem in software product engineering: Software products are highly complex, technical, and require much greater "inbound marketing" response—altering the product based on customer needs—compared with traditional products such as soap, candles, and toothpaste. This need sparked the creation of a role less bound to Sales and Marketing, and more to Development or Engineering: the role of Program Manager, discussed next.

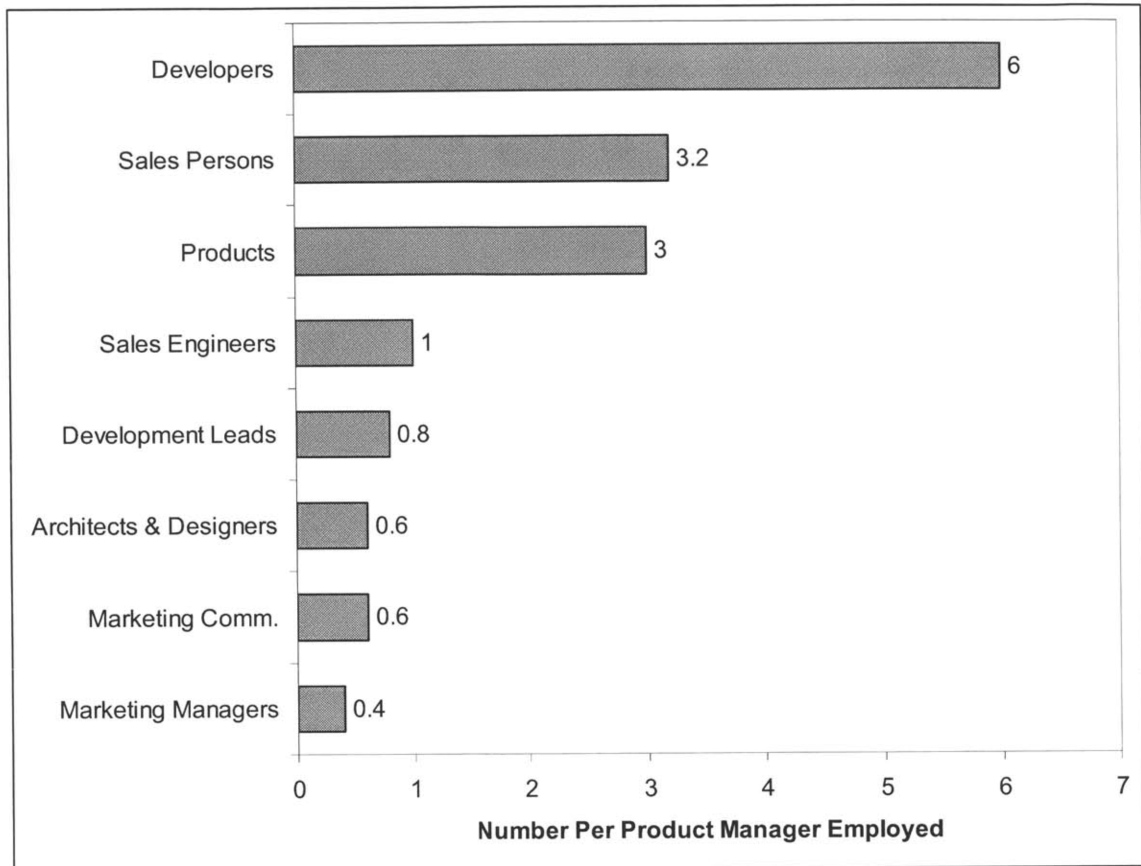


Figure 5 Product Manager Employee Ratios (Data from Pragmatic Marketing) [13]

Program Management

Microsoft realized in the 1980's that the job of Product Manager in high technology products, including all pure software products, did not have a one-to-one correspondence with the Proctor & Gamble role. They realized that in high technology, Product Management spills over from a marketing function into being partly an engineering function as well. As Proctor & Gamble pioneered the role and job of Product Manager, so Microsoft pioneered the role and job of *Program Manager.* As Cusumano & Selby note in *Microsoft Secrets,*

> The first turning point was a decision in 1984 to set up testing groups separate from development...The second, occurring about the same time,

was when program management began to emerge as a function distinct from product management and software development.[9, p. 36]

Later, a Microsoft program manager, Bruce Ryan, is quoted giving the key responsibilities for program managers:

- the product's vision
- the written product specification
- the product schedule
- the product development process
- all implementation trade-offs
- coordination of the product development groups [9, p. 77]

Contrast this with Microsoft's description of Product Managers as "'MBAs' and 'snappy dressers [who] own their own homes" and have five areas of responsibility:

- oversee a "business"
- recognize and pursue market opportunities
- aggressively represent the customer in the product development process
- take responsibility for the trade-off between functionality and ship date
- take responsibility for the marketing and sales process [9, p. 89]

As of this writing, the Microsoft site describes the role of Technical Program Manager as follows:

Driven to Succeed

Program Managers are customer focused, working to ensure that the products Microsoft produces will delight users and enable them to do their best. Program management is also an opportunity to flex technical muscles: your technical decisions and direction are what drive products and features through to completion.

Working across multiple groups with marketing and sales personnel on the customer end, program managers translate customer requirements into product features and create functional specifications. On the

> implementation end, they prioritize and deliver on those features, working closely with key technical resources, such as software development, testing, documentation, localization, tech support, and more.
>
> Program Managers typically have a software development background. This technical expertise is blended with evangelism, empathy, conflict negotiation skills, and a passion for driving projects through to completion. [14]

According to Microsoft academic liaison and former Product Manager Philip DesAutels, Program Managers can be quite influential within Microsoft and become more managerial and less technical at senior levels if they are involved with a large, vital product suite such as Office. The title Program Manager is also somewhat overloaded to some degree and can apply to a variety of different specialties. For instance, the Senior Standards Program Manager of Corporate Standards is categorized as a "Legal and Corporate Affairs" position.

The Program Management Grey Area

Earlier, we noted that Product Management is a somewhat difficult role to pin down and define precisely. Program Management suffers from some of the same ambiguities. This difficulty percolates up to recruiting issues:

> There is no particular university degree that qualifies someone for a program manager's job....Gates himself observed: "Program management is weird, because where do you recruit program managers? What is the background for a program manager?" [9, p. 96]

Microsoft Secrets defines Program Managers as follows:

> Program Managers have the broadest and most ill-defined set of responsibilities...[they] tend to have backgrounds that combine a strong

> interest in design issues with some knowledge of or familiarity with computer programming. [9, p. 97]

Yet a very different version of the role comes up in his newer book, *The Business of Software:*

> If managers of new projects wanted to move *really fast*...developers usually took the lead in proposing features and writing up specification outlines. In these cases, Microsoft program managers came on board later and worked mainly on managing project schedules, writing up test cases with testers in parallel with development, working with interface or Web page designers, and *building relationships with outside partners and customers*.[15, p. 160, italics mine]

This last—building outside relationships—sounds a lot like the role of a Product Manager in serving as the customer's voice. Thus, the roles can overlap in some cases. The fact is that both roles lay under the general umbrella of middle management and both act as glue between what software engineers are actually implementing and what the business is trying to accomplish. Most large software product companies today have both Product and Program Managers, although startup- and medium-sized companies often do not have Program Managers. In this case, the job of Program Management is split between Development and Product Management. Virtually all software product companies of any reasonable size do have Product Managers.

Conclusions on Product and Program Management

In general, the Program Manager position is a technical and feature-specific permutation of the Product Management role. Created by Microsoft, the role has been adopted by many other software companies. The overlap between the Product and Program Management roles is as shown in Figure 6: both roles involve some two-way

interaction between Engineering and Marketing, but Program Management is more deeply embedded in and tightly bound to Engineering and product feature details, while Product Management is more deeply embedded in Marketing. A Product Manager would be more likely to prioritize features based on the potential added sales and market value of the product, whereas a Program Manager would be thinking a lot more about how to integrate specific features into the interface, and how to prioritize them based on ease or difficulty of implementation.

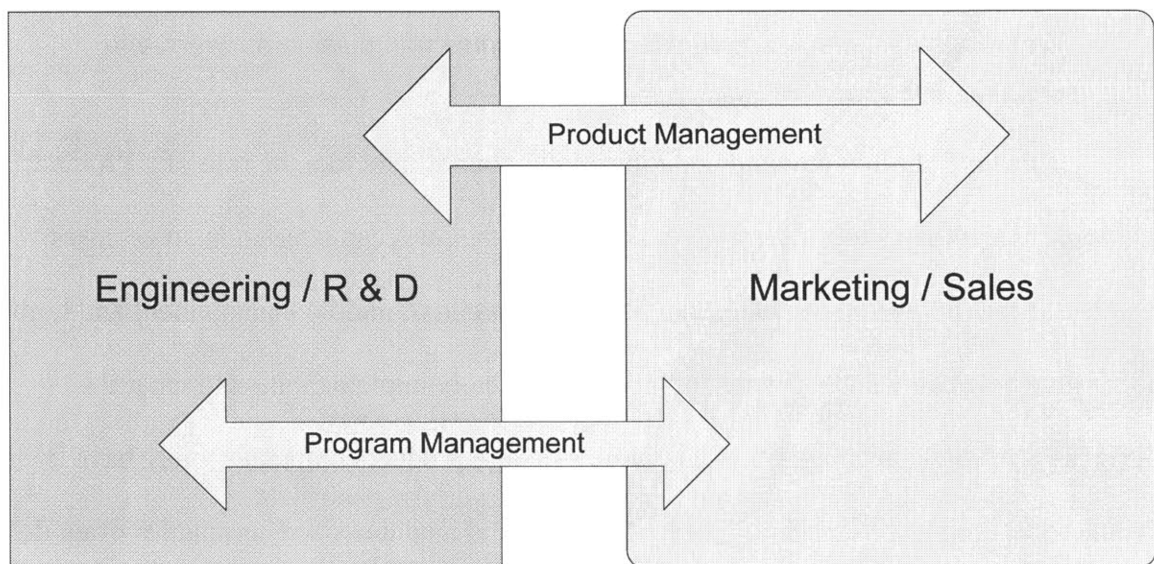


Figure 6 Relative Engineering / Marketing Overlap of Product / Program Management

This relationship seems to hold for Program Management positions in non-Microsoft companies. For instance, Scott Case noted in his interview:

> The Product Manager decides what features to add to the product. He or she will work with one or two Program Managers to prioritize the features, and will write a Business Requirements Document to give an idea what should generally be in the release. The Program Managers work with developers and Project Managers to kick back schedule estimates. As needed, of course, there is a feedback loop to clarify and change course during a project.

The role of Product Manager is perhaps somewhat more *strategic* where as Program Manager is more *tactical*, although flipping of this relationship is possible in a company as large as Microsoft, in which the job of Program Manager has considerable prestige. Bill Shelton claimed some coworkers were surprised when he moved to change his role from Program to Product Manager, viewing it as a lateral move or even a slight slip in prestige. There is a certain intangible prestige to being technical at Microsoft, and Cusumano has noted that developers "walk taller" than other employees in the company, which has formal and quite well-paying technical career tracks. Both roles share the defining responsibility of cross functional team coordination to achieve technical and market success of the product.

IV. Struggle and Failure in Software Product Management

An Extended Case Study: Netscape Navigator versus Internet Explorer

In the military, one strategy to train both troops and their strategic commanders is to play war-games. Past battles are also studied in detail by military strategists. The purpose of this section is to view Product Management from a commercial war-games or combat retrospective viewpoint. The moves, in some cases, were made by CEOs and not directly by Product Managers, but PMs face precisely analogous strategic decisions, although usually on a smaller and less public scale. One aspect of Product Management this section illuminates, which we have not previously explored, is the importance of company culture in the treatment of partners and customer-partners. PMs are often on the front lines of these relationships and have a lot of influence over the tone of a company's relationship with its partners.

One major motivation for this thesis was the collective works of business historian Michael Cusumano, author of several books on software product innovation as it relates to actual business value. The particular commercial combat retrospective we use is derived from his book, *Competing on Internet Time,* co-authored by David B. Yoffie. We attempt to extract the most poignant nuggets of wisdom from its over 300 pages with the advantage of writing with historical perspective in 2006, the Browser Wars long past.

A lot of luck was involved in the creation of Netscape, and highly touted genius Mark Andreessen stood, like most inventors, on the shoulders of giants. One giant in

particular being Tim Berners-Lee who, for all intents and purposes, created the Web at CERN. Andreessen created NCSA mosaic on top of the Berners-Lee-designed infrastructure, and eventually became a founder of Netscape and its Netscape Navigator browser. Navigator would eventually evolve into a suite of bundled applications and be rechristened Communicator, but more on that later. Cusumano and Yoffie cover the creation of the Netscape company in detail, and we will not discuss it in detail here. Pure product management does not exist as an independent role until a company has reached a certain critical mass and level of business infrastructure. Certainly Andreessen and co-founder Jim Clark thought about aspects of the business we would consider product management, but we are interested in businesses that have evolved to the level of being able to support product management as an independent role.

Of course, Netscape evolved to this level very quickly. It is impossible to track in any detail what the particular Product Managers within the company were doing, but we can look at the entire company's product management policies based on the results of their technology and partner initiatives. *Competing on Internet Time* also provides a few peeks at what was going on with Netscape's product management internals. A high level look follows at the lessons and ideas learned from Netscape's actions in Product Management, both in capturing an open market and, later, in reaction to Microsoft's entry into the browser market.

Company Culture Matters: Attitude is Part of the Product

Over the long haul, people problems tend to dominate technology problems in any product with a mass market. In fact, most professors in MIT's Engineering Systems Division seem to feel that human cultural problems dominate technology problems,

period. Everything seems to point to NASA's cultural problems in the space shuttle accidents, for instance. Netscape's company culture certainly had its issues, especially as a business partner. Netscape was young, Netscape was hot, Netscape was a Wall Street darling, and Netscape was arrogant.

Cusumano and Yoffie note, "The perception of arrogance was widespread. This perception emerged fairly early in the company's history." An extended quotation from Michael Dell, chairman of the now-dominant Dell Computer, is also given on this subject:

> Netscape was surprisingly arrogant for a company of their size and age and didn't seem to aggressively pursue our business. It was just a number of little things that sent the wrong signal to us. They didn't appear to engage us very heavily in pursuing our business. And...Netscape started to align itself with Oracle, IBM, and Sun to support the network computer phenomenon, even though their bread and butter counted on the PC. [16, p. 85]

Karen Richardson, who came on board Netscape as VP of strategic accounts, was also quoted on the arrogance topic:

> A lot of the arrogance had to do with the way partners, customers, and potential partners were treated. Everything from do you show up to a meeting on time to attitude. [16, p. 83]

Everyone knows the old adage, quoted to the point of cliché, that "power corrupts, and absolute power corrupts absolutely." Microsoft is certainly not known as a humble company and has oft been reviled for its perceived arrogance. Microsoft has been market dominant to the point of monopoly and Netscape in the mid-1990's had quickly achieved (what would turn out to be shockingly temporary) market dominance. Perhaps it is true

that "market power corrupts, and absolute market power corrupts absolutely" when it comes to a company's attitude and culture. Success breeds arrogance.

The problem with arrogance is that it causes incredible resentment in those on the receiving end. In business, a company always needs its partners and customers to behave in an economically rational fashion, or, through marketing, branding, image, and strategy, to cause partners and customers to behave irrationally in the company's favor. Arrogance can easily cause strategic partners and customers to behave in an economically irrational way *against* the company: purchasing products or partnering with someone else even if the arrogant company genuinely has a superior economic value proposition. Obviously, this can have catastrophic long term impact on profitability. To make matters worse, extreme arrogance, such as that exhibited by Microsoft, can radicalize a company's enemies and galvanize them.

Arrogance at Microsoft

For instance, there is a large contingency of "disaffected software developers," users, businessmen, and competitors, which Cusumano and Yoffie refer to as the ABM crowd: "Anybody But Microsoft." There is strong anecdotal evidence that Microsoft's corporate arrogance along several dimensions has created this crowd, or at least made it much larger and more cohesive than it would have been had Microsoft shown a touch of humility or taken a more nurturing role. One might even speculate that the rapid spread and adoption of Linux may have been significantly accelerated by deep resentment of Gates & Company's domineering arrogance. Of course, with such a strong monopoly, Microsoft has withstood a lot of arrogance. On the other hand, its growth on the stock market and influence in the world have reached a plateau in recent years. How much of

this can be attributed to the company's arrogance creating a vast horde of Microsoft haters, all feeling like a part of *La Resistance* and often willing to go against their own economic interests if they may bring down the arrogant giant over the long haul?

Arrogance at Netscape & "Mooning the Giant"

Netscape lacked Microsoft's monopolistic position and its arrogance may have hurt the company much more deeply in the short term. Imagine the egregious magnitude of the strategic error in coming across as arrogant to Michael Dell and failing to form a partnership. Conversely, think of what might have happened if Netscape had aggressively pursued Dell, played on Michael's fears of Microsoft's OS dominance controlling his PC platform, and convinced him to bundle Netscape with every Dell PC by the late 1990's? The market proved Dell did not need Netscape, but the outcome of the Browser Wars might have been quite different had Netscape become tightly aligned with Dell.

According to Christian mythology, "Pride goeth before destruction, and a haughty spirit before a fall."[17] Pride, or arrogance, may not destroy your business in the short term, but it will surely make your competitors and perhaps even what could have been your potential partners stick their feet out to trip you and see your business fall!

Company culture, it has been said, comes down from the top. A culture of arrogance can certainly come down that way. In a case of arrogance versus arrogance, Clark & Andreessen took a very poorly advised stance of arrogant defiance of Microsoft, which *Competing on Internet Time* refers to as "mooning the giant." Clark repeatedly referred to Microsoft as the "Death Star" and asked for trouble by claiming that "Netscape was developing a full-fledged networked operating system that would make

Windows unnecessary and outdated." Meanwhile, Andreessen was quoted as saying that Netscape technology would relegate Microsoft's OS to being "a mundane collection of not entirely debugged device drivers."[16, p. 105] David may have taken down Goliath with a well-aimed slingshot bullet, but Andreessen's PR strategy was akin to simply hurling insults at the giant. The giant was more than capable of crushing David in this scenario, and, of course, Microsoft went on to make mincemeat of Netscape's market share.

Regardless, Clark and Andreessen's arrogant attitude toward Microsoft may have encouraged company arrogance in general. Netscape lacked what might be called strategic humility. In managing even a very successful software product, it makes sense to always be respectful and reasonably humble in dealing with partners, potential partners, and customers—even those who may not seem important in the short term.

Judo Strategy

Cusumano and Yoffie discuss Netscape's competitive strategy in the chapter of the same name, focusing on Judo as a driving metaphor. Much of the book references this metaphor, and it is worth repeating the four fundamental tenants of struggle, which applies both to company and product management:

- *Move rapidly to uncontested ground in order to avoid head-to-head combat.*
- *Be flexible and give way when attacked directly by superior force.*
- *Exploit leverage that uses the weight and strategy of opponents against them.*
- *Avoid sumo competitions, unless you have the strength to overpower your opponent.* [16, p. 90]

We will touch on each one of these concepts. For coverage in depth, the book is an excellent case study for any software Product Manager to read, especially in the brave

new world in which nearly all new enterprise and consumer software products are either Internet-centric or at least Internet-enabled.

Uncontested Ground

"Shoot, Move, and Communicate"

This quote from former Netscape CEO Jim Barksdale, taken from the marine infantry's standing orders, reflects his philosophy of having Netscape be a nimble business combatant against the larger but less flexible Microsoft. The idea is to avoid getting pinned down by a competitor in a rough head-on battle, such as a direct firefight against Microsoft. Of course, as seen above with the "mooning the giant" tendency Netscape had, they were not always consistent with this philosophy. However, they did outflank and outmaneuver the competition in browser space, and beat out several smaller competitors. If the first volley of rounds was creating and releasing the browser itself, the Netscape platoon quickly moved to fire at the competition from a different angle: they took greater advantage of the online ability to download the browser than did the competition.

"Free, but not free"

Netscape continuously altered the rules of the market for competitive advantage. Product pricing is an important part of PM, and Netscape fired at competitors from the pricing position with an innovative pricing strategy: "Netscape browsers were free for anyone to download on a 90-day trial basis, free for students and educational institutions, and $39 (later raised to $49) for everyone else."[16, p. 99] By contrast, one competitor named Spry, kept "firing" from their position on retail shelves, convinced that the old

retail model and dial up Internet was a solid position from which to fight a war of attrition. Spry had an early entrant position over Netscape, but their lack of maneuverability and continued focus on shrink-wrap proved a tragic mistake in Product Management for them.

Limits of Movement

Moving too much in search of uncontested territory "can confuse customers and undermine a company's strategic credibility." It can "look like inconsistency…lack of focus and a lack of commitment." Andy Grove, Intel's chairman, said,

> The battle between Microsoft and Netscape can be described as a guerilla war against an occupying army. Netscape originally was going after browsers, then going after consumers, and then they changed their strategy to corporations…the problem is, they're running out of space, munitions, and food.

The idea was that the constant strategy changes were raising serious questions about Netscape's ability to commit to a strategy and execute. This further jeopardized a company already shaky in the area of partner relationships. To close this discussion, we will quote former Netscape employee Alex Edelstein, who told Cusumano and Yoffie,

> I'm going to strangle the next person who tells me, "We have to change the rules, Alex, that's the only way we are going to beat these guys." Because that is a very valuable tool, but you cannot use it as a crutch, as a replacement, as a surrogate for execution.[16, p. 104]

Be Flexible

Aesop's "Fable of the Oak and the Reeds" epitomizes the Judo strategy of being flexible, elucidated to members of Generation X in the pop culture by the character of

Paul Atreides (played by Kyle MacLachlan) in Frank Herbert's "Dune", who thinks (aloud via voiceover), "I will bend like a reed in the wind!"

For better or for worse, the best and most shocking example of flexibility in the Browser Wars came not from Netscape but from Microsoft. Everyone in the Silicon Valley eventually heard about Gates' famous memo from 1995, "The Internet Tidal Wave." In a major break from previous strategy, Gates proclaimed, "Now I assign the Internet the *highest* level of importance. The Internet is the most important single development to come along since the IBM PC was introduced in 1981." [16, p. 108]

Microsoft's Master Stroke of Framing and Flexibility

Flexibility

It is easy to say, in retrospect, that any fool could see that the Internet as a non-proprietary platform would rule the world by 1995, but this is not necessarily true. Microsoft had massive resources and power, and an economic interest in the proprietary Microsoft Network, which Gates essentially threw over a cliff when he decided to use an *embrace and extend* strategy for the Net. At a very high level, this may have been one of the single most brilliant, or at least economically most important, Product Management decisions in the history of software. In terms of visionary accuracy, the insight behind this decision was perhaps only superseded by Gates' original notion in the 1970's that the software, not the hardware, was the key product. Within a week after Gates said, "We're hard-core about the Internet," Netscape's valuation had fallen by 28 percent.

Framing

Beyond flexibility, Gates' speech was brilliant from a public relations standpoint. They chose December 7th, the anniversary of the bombing of Pearl Harbor, as the date to

speak with the press. In the Harvard Business School experimental course, *Strategic Reasoning Laboratory*, Professor Giovanni Gavetti teaches about the importance of framing a problem in terms an audience will understand, and in terms that the audience will not resist. Often viewed as the "Evil Empire" or a big corporate bully, Microsoft has not been a master of good PR. However, on that day in 1995, Bill Gates gave a masterful analogy:

> I realized this morning that December 7th is kind of a famous day…And I was trying to think if there were any parallels to what was going on here…the most intelligent comment that was made on that day wasn't made on Wall Street, or even by any…analyst; it was actually Admiral Yamamoto, who observed that he feared they had awakened a sleeping giant. [16, p. 109]

This analogy triggers the "frame", in the Gavetti sense, of Microsoft not as a bully, but as a powerful yet heroic and patriotic giant being hurt unawares by an angry and aggressive, but weaker upstart. All but Microsoft's most adamant enemies could not help feeling a vague sense of patriotic sympathy with Microsoft on hearing the Netscape / Microsoft battle framed in this way. Even dyed-in-the-wool Microsoft haters could not avoid having the frame triggered and a momentary confusion of feelings about the Evil Empire: that is just how the human brain works.

Gavetti noted in his class that all of HBS, driven by the case method, is based on analogy. Framing is critical in Product Management as elsewhere in business. The most important reading for the class, in terms of framing, was a book by George Lakoff, *Don't Think of an Elephant.* (This is unrelated to the Blind Men and the Elephant parable discussed earlier.) Every PM who faces framing product and company issues for the press should read the first 35 pages of this book. As an example of the power of framing,

Lakoff notes how important it is to have a good frame and how even attacking an opponent's frame can work against you:

> When we negate a frame, we evoke the frame. Richard Nixon found that out the hard way. While under pressure to resign during the Watergate scandal, Nixon...stood before the nation and said, "I am not a crook." And everybody thought about him as a crook. [18]

Most Product Managers must communicate with the press: never underestimate the power of framing!

Exploit Leverage

Cross Platform versus Planned Obsolescence

Netscape found weaknesses in Microsoft's strategy in that the larger company had a large installed base of legacy products. Microsoft relied partially on a regimen of "planned obsolescence" and its new Internet Explorer product would not immediately run on PCs with older Windows versions in many corporations. IT managers in companies tend to upgrade much less often than home users for reasons of cost and operations headaches. Barksdale exploited this by speaking of the "UNIXification of Windows" and Netscape took the strategy of supporting the whole installed PC base.

This was, temporarily, an effective strategy and slowed down Netscape's browser share loss. However, as Cusumano and Yoffie point out, "The weakness of Netscape's strategy is that the half-life of DOS and Windows 3.1 is finite: Over time more and more companies will migrate...*points of leverage do not last forever, especially when competing on Internet time.*"

Netscape also attempted to use cross-platform technology as leverage for their products, especially Java technologies. Product Managers beware, though: Going cross-

platform always sounds good when going up against a highly proprietary competitor, but has extremely high hidden technical cost. "Cross-platform products pose technical challenges that can lead to lower programming productivity and weak product performance compared to platform-specific products." Netscape never did do a successful complete rewrite in Java. Bob Lisbonne, Netscape's V.P. of the Client Product Division, said,

> The 6.0 project, which was code-named Xena and was basically a new Communicator [Netscape's 2nd generation browser] all written in Java, has been shelved. And that's been shelved primarily for technical reasons. The reality was that Java...was just not yet up to the task of implementing a product of that complexity. [16, p. 193]

Mark Andreessen himself said, "I just didn't kill it soon enough....And 6.0 turned into rocket science, and it was driving me nuts." An in depth study is beyond our scope here, but in the author's experience, planned (or fortuitous) obsolescence strategies have dominated over pure cross platform and backward compatible approaches in software and hardware product histories, at least in terms of profitability.

"Open, But Not Open"

In terms of open interfaces and Internet protocols, Netscape did a fairly good job of exploiting leverage and framing to paint Microsoft as the evil proprietary alternative and themselves as the "guardian of greater 'openness'". This strategy was intended to appeal to what was known in the Silicon Valley as the ABM crowd, who wanted Anybody But Microsoft to win any technology war and wanted to use products from, well, Anybody But Microsoft. This was good positioning, or framing as Professor Gavetti would say, but behind the PR framing lay the truth that Netscape would offer

"very subtle features that are proprietary or difficult to copy." In one of the more important insights of *Competing on Internet Time*, the following "open secret" is revealed:

> In fact, the dirty secret of the computer industry is that everyone is "open, but not open"; they differ only in degree. Every computer company has proprietary pieces in its solutions, while every company in the industry claims to be "open," including Microsoft and IBM. [16, p. 133]

There are two kinds of frames: Frames that match the underlying facts, but creatively, and frames that don't. The latter kind can be powerful and manipulative toward a desired end, but are also subject to backfiring. The reality of Netscape's "open, but not open" strategy became apparent and slowly shifted customer perception to blur the distinction between their "champion of openness" stance and the perception of Microsoft as highly proprietary. Product Managers should beware of risking their integrity with customers by using frames which do not match the facts! Fact-matching frames are like white magic: powerful, positive, and rarely troublesome. Disinformation frames are like black magic, even more powerful for manipulation, but prone to turning negative and, depending on the level of the mismatch, extremely dangerous to a product's (or suite of products', or a company's) long term reputation.

Overpowering with Sumo Competition

Cusumano and Yoffie give Microsoft credit for one more judo move, in giving away Internet Explorer. This goes too far in giving Microsoft kudos for its judo. The ultimate market-crushing hammer of the incumbent heavy in any industry is to drop prices to squeeze out a new competitor. One of a myriad of possible examples is the Harvard Business case dealing with Braniff versus Southwest Airlines: essentially,

Braniff tried to crush Southwest Airlines through pure price wars, and Southwest responded with a truly brilliant judo move. Southwest framed Braniff as the big price war bully, appealing to the public's sense of fairness. The exact move will not be revealed here lest the HBS case mafia comes after the author, but the move was true business judo, and it worked.

The point is, *Competing on Internet Time* unfairly claims, "Microsoft was using Netscape's weight against it. Netscape became caught in a classic judo move." The move was classic, all right, but it was a classic bullying tactic by a heavyweight opponent with superior capital and an operating system monopoly. From later in the book: "Netscape management seemed immobilized by Microsoft's 'free IE' judo move." Again, Netscape may have been immobilized, but clearly by a *sumo* move, not by a judo move. Judo has nothing to do with Microsoft's brute force attack on Netscape's market share by using Internet Explorer as the ultimate Internet loss-leader product: a browser given away free. The Silicon Valley watched in horrified fascination as Microsoft assaulted Netscape with this ultimate sumo attack, pushing against the boundaries of business ethics: was Microsoft leveraging one monopoly to gain another?

History seems to support our dissenting view. Cusumano and Yoffie themselves quote one Microsoft representative as saying in public, "Our intent is to flood the market with free Internet software and squeeze Netscape until they run out of cash." They do acknowledge at the end of *Competing on Internet Time,*

> But Gates and company were too greedy and too tough when it came to winning market share in the browser wars. In winner-take-all environments, firms can gain so much market power and market share that they have special obligations under antitrust laws...you cannot use your monopoly power to hurt a competitor in another market. [16, p. 319]

Why Microsoft is given credit for a judo move here, when giving away Internet Explorer was clearly a *sumo* move at best, and an illegal monopoly-leveraging sumo move at worst, remains puzzling. Perhaps it is an example of being seduced by one's own frame: the driving frame of the book is the judo contest. In an effort to remain balanced, trying not to paint Microsoft as a sumo-centric bully, Cusumano and Yoffie perhaps tried to fit more of Microsoft's strategy into the judo frame than actually merited by the business colossus of Redmond.

Marketing Warfare in Product Management

It is evident from the Netscape / Microsoft browser wars that all "four P's" of marketing—Product, Price, Positioning, and Promotion (they had to come up somewhere in a work involving marketing)—come strongly into play for modern Software Product Management, particularly in a contentious space. Netscape waltzed into an open market with an excellent new *product*, startling Microsoft, which had left open a market power vacuum. Microsoft spent egregious sums creating their own product, using an essentially unlimited budget and quickly coding up what turned out to be a solid, modular, decent alternative. Both companies had compelling products. Without a compelling product, no amount of Product Management can make a company a market winner beyond a very short "hype window".

Netscape competed on *price* with a unique new "free but not free" model also involving positioning, but their market was eventually destroyed when Microsoft detonated the thermonuclear pricing war device of giving away their version of an equivalent product. *Positioning* Netscape Communicator as a downloadable solution helped Netscape win over Spry in the short term, and they later used positioning judo

through a cross platform strategy, compatible with more versions of Windows than Microsoft's own Internet Explorer. Judo moves only are effective for a few seconds of market time, and eventually Microsoft's brand name and possibly illegal arm-twisting of PC clone makers effectively steamrollered over Netscape for positioning dominance.

Of course, both companies madly *promoted* their product, with Netscape benefiting from first mover advantage, a much-hyped IPO, and the executive celebrity duo of Andreessen and Clark. Bill Gates has worldwide, long lasting celebrity, also, and Microsoft's market dominance of the operating system allowed them to win the key promotion battle for icons on the Windows desktop. The battle was truly an epic one crossing business and legal boundaries, with Microsoft ultimately the victor, but also ending up in court for antitrust violations. Product Managers, especially those with similar outbound responsibilities, can expect to fight similar battles in their careers. There is no formula for winning these battles, but shrewd PMs will focus first on having excellent product, strategize on all "four P's", and keep an eye out for leverage opportunities using marketing judo.

V. Software Project Management

Product and Program Management are distinct roles from Project (or Development) management in most software companies, but neither is completely insulated from the details of project management. Program Managers are intimately involved in the tactical details of developing new features, including working with Development Managers on project schedules to decide what feature set can be developed for new product versions. As for Product Managers, 79% surveyed in 2005 reported the responsibility of "monitoring development projects" as part of their jobs. [13] Thus, a study of project management methodology is necessary to understand fully the role of the Product and Program Manager in relation to the software development team. Since plenty of literature exists on project management, we will only survey the major methods in use today, briefly critiquing how each might affect software architecture and company culture.

Waterfall Model

The first traditional software development methodology, the Waterfall Model, is linear and inflexible. In the unlikely event a Product Manager is dealing with a pure Waterfall Model shop, it will be extremely important to define all requirements with absolute precision and an ironclad contract of execution. The Waterfall Model (Figure 7) can be described as "Big Bang" project management: the system and software requirements are the seed of the software product universe, and six months to two years later explode into a perfectly completed software product. This generally does not happen, as inevitably upstream changes are required. It is extremely expensive to make

"upstream changes" in this model. Probably the U.S. Department of Defense is one of the only organizations left using this model in its purest form, and even they are beginning to use more iterative methods in new projects.

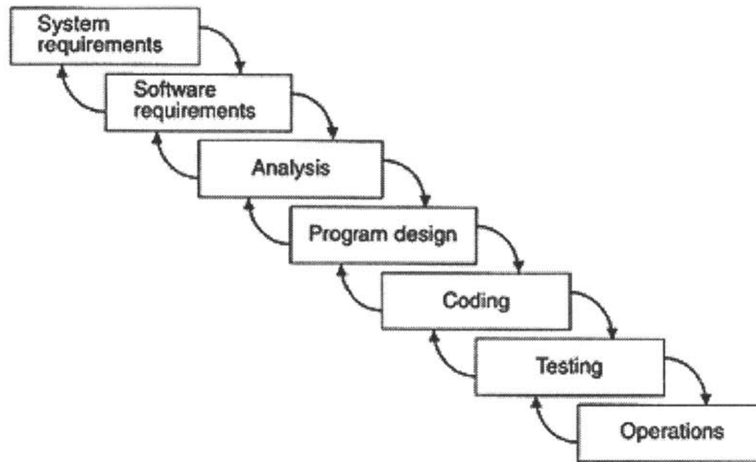


Figure 7 Waterfall Model (Source: http://www.serverpeak.com/images/waterfall.gif)

Spiral Model

The Spiral Model (Figure 8) is perhaps the most important technological innovation for modeling the software process after the Waterfall Model, which is simply not sufficient to capture the need for the many iterations necessary to produce most software products. Most modern software project management methodologies derive from the Spiral Model. One of the main differences between modern methods is how "tight" the turns of the spiral are. A sync-and-stabilize company like Microsoft might have spirals that revolve in a matter of weeks, whereas some of the more *avant-garde* methods can undergo an entire turn in a day or possibly even a fraction of a day.

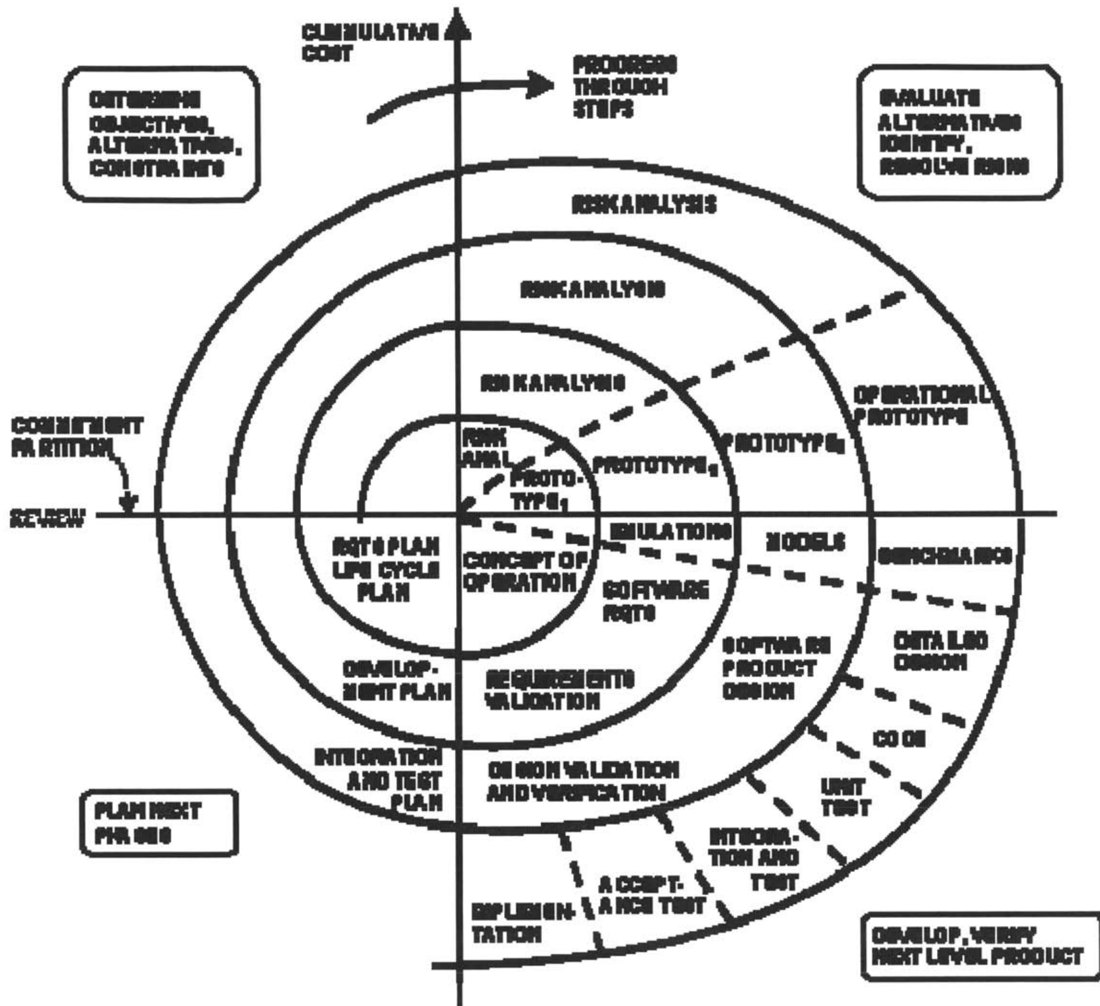


Figure 8 Spiral Model (Source: http://sunnyday.mit.edu/16.355/spiral-model.gif)

Some of these newer methods are sort of "spiral waterfalls", and it is possible to combine the two methodologies by creating bite-sized linear project sub-plans and then spiraling them together to allow for iterations. Extreme Programming is a good approximation of this combination (see below), with little two week waterfalls, but the collection of waterfall model processes is more iterative and spiral in nature.

Not all phases shown in the figure are used in all versions of the Spiral Model in industrial use. The common feature is an iterative approach. The important note for the Software Product Manager is that this model involves creating successive prototypes, or

incremental releases, of the product. Thus it is much easier to inject requirements changes into this process as customer requirements evolve or are better understood, because the spiral will swing around to a new level of requirements input and validation. This is no excuse for not carefully formulating requirements at the beginning of a project, however, and repeatedly injecting serious requirements changes at all angles of the spiral is probably the most efficient way to drastically reduce or even eliminate a project's chances of success.

Agile Software Development

Agile Software Development sounds like a consulting company but, according to one article in *Computer* by IEEE, it is actually a collective term for multiple new project management methods for software including Extreme Programming, Crystal Methods, Lean Development, Scrum, and Adaptive Software Development. All of them are modifications of the Spiral Model. These methods, called *agile methods*, are good news for a Product Manager with changing requirements from customers. In today's business world, eliminating change early in the development process causes unresponsiveness to business concerns and therefore causes PM difficulties. Reducing the cost of reactive change throughout the software project is the goal of agile methods. Working code is emphasized over documentation, and there are several other preferred methods emphasizing individuals, interactions, customer collaboration, and change responsiveness. Agile methods don't involve inclusive rules but generative rules, a "minimum set" of behaviors to devise appropriate practices for special situations. Intense interactions between team members are emphasized. The goal is to produce software that

actually meets customer requirements and workplaces that "aren't described in Dilbert cartoons." [19]

Extreme Programming (or XP)

Extreme Programming is perhaps the most well known and fashionable of the new wave of agile methods as of this writing in 2006. One introductory article by Robert C. Martin elucidates the basic tenet of Extreme Programming: frequent feedback. [20] The "Two Week Plan" is at the core of this model. Instead of a six month software product plan which forms one large waterfall model of development, XP is more spiral-like in having "mini-waterfall" plans of two weeks in duration. Then all these mini-waterfalls are spiraled together: the approach is to incorporate the feedback from the past two weeks into the next two weeks, and iterate.

Some tenants of XP, such as the requirement to have all code literally written with two people at a desk, seem impractical. This policy goes completely against the solitary nature of programmers, who like to go off on their own to do the "real work". Since very few programmers have exactly the same skill level, speed, and style of hacking, this two-to-a-desk model—if forced—would most often degenerate into a tutorial / training session for one programmer and a drain / annoyance on the more advanced programmer. The two-to-a-desk model may sound good in a Project Management book, but rarely happens in industry due to cost and personality issues. However, the basic idea of reducing the trust (or risk) horizon to two weeks instead of six months is a good one. The spirit of Extreme Programming is to try to increase communication and build trust by delivering features in smaller time increments. A Product or Program Manager operating in a corporation using some variant of XP has the advantage of being able to inject

requirements modifications quite often. This is helpful, but there is a dark side: this method lends itself to drifting away from the original requirements as programmers and Project Managers continually revisit requirements and may have a tendency to dumb down those requirements which are hard to implement, while expanding on features that are easier to implement. Thus, in this model, the Product Manager will incur more communication effort in keeping the development teams on target with original requirements, but will also find teams to be highly responsive to needed changes.

Scrum

Scrum is perhaps right behind XP in popularity. According to the home page for Scrum---http://www.controlledchaos.com—Scrum is "A variation on Sashimi, an 'all at once' approach to software engineering. Both Scrum and Sashimi are suited best to new product development rather than extended software development." Figure 9 gives a rough depiction of the development cycle. Interestingly, some companies do use Scrum for continued development.

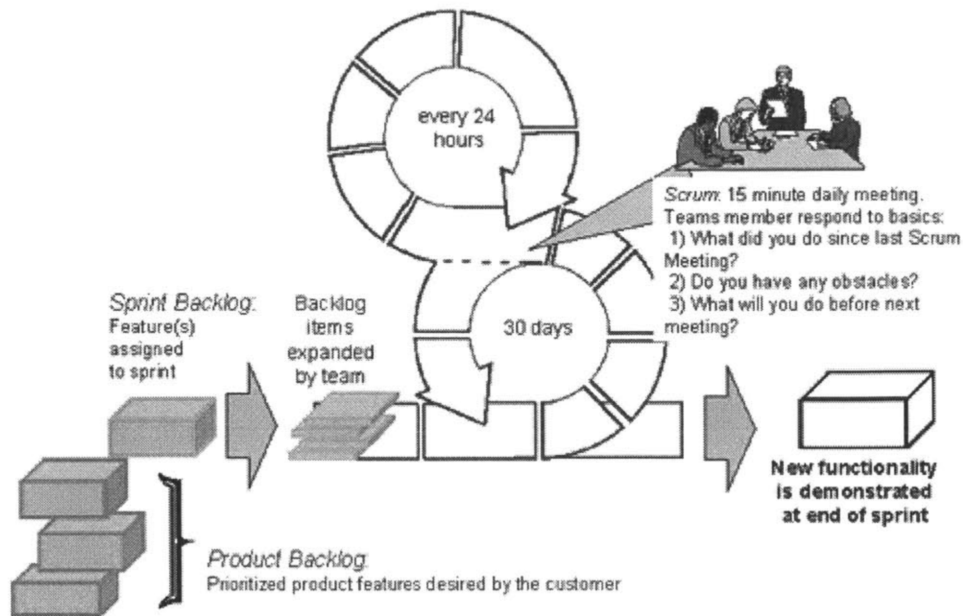


Figure 9 Scrum development cycle (Source: http://www.controlchaos.com/)

Scott Case, a Product Manager interviewed for this thesis, works for Atlas Solutions in the Internet advertising industry. The core software system serves ad banners. "Development leads use Scrum methodology, and work together," he said. "Scrum meetings are internal, but then there is a once-a-day 15 minute meeting of team leads."

The People Factor

One article by Cockburn & Highsmith advocating agile methods emphasizes the importance of people in agile software development. Placing people physically closer, replacing documents with whiteboard discussions, and improving the team's "amicability" are all encouraged. Important summary slogans include "people trump process" and "politics trump people"—people of talent can work their way through any process, but process will not make incompetent people perform; and corporate politics,

including inadequate executive support, can kill any project regardless of the talent of the front line people involved.[21]

The issue with these so-called agile development methods is that they seem to have been created in a psychological vacuum. Programmers are private people. They are willing to be a little closer, spend a bit more time in meetings, but they view it as punishment to be forced to sit in a group all day long. Some ideas from agile methods make sense, but others will be proven by history to be the outcome of a 5-7 year "manic episode" in the history of software development. For instance, the Highsmith & Cockburn article (published in 2001 and probably written before the true catastrophic nature of the crash was fully apparent) states, "replace documents with talking in person and at whiteboards." This is ridiculous. People are going to talk and use whiteboards anyway in a functional organization, and it might make sense to *reduce* documentation, but it is ludicrous to ditch all documentation. Organizational memory vanishes in this scenario. This kind of attitude was indicative of the times: "We don't have time for documentation!" was the prevailing attitude, as over-funded startups snorted the free flowing metaphorical cocaine of excessive Venture Capital and made sloppy rushes to market.

Fashionable Agile Methods: Programmers as Cattle?

Agile methods have a catch: some of the ideas are repugnant to talented programmers. For instance, the very idea of forcing software engineers, who tend to be thoughtful, highly intelligent individualists, into "barrier-free collocated teams" might sound good on paper but totally ignores the psychology of what it means to decide to become a software engineer. Many startup companies got ahold of these agile methods

and used the "barrier free" idea to take away any privacy a given programmer might have at work. Programmers were forced, like pigs in a pen, to sit within smelling distance of each other at long tables—sometimes no bigger than card tables, and in the worst of the dot coms, *actually* card tables—all in the name of some project management fad. To many who experienced this methodology as software engineers, including the author, the open area "bullpen" concept came across as a lame excuse by management to deprive workers of personal space and privacy. The policy created a Dilbert-esque, oppressive office atmosphere. Abusive aspects of these project management fads ran rampant in many companies which failed. Even Morgan Stanley Dean Witter Online, which used a bastardized version of XP, no longer exists today. It is ironic that the Cockburn & Highsmith article subtitled *The People Factor*, shows a complete neglect and ignorance of the psychology of the workers performing the projects. [21]

Many if not most software engineers are uncomfortable in large groups for long periods of time: denying them personal space at work is tantamount to punishment. Promoting greater levels of interaction is important, but Product Managers should be prepared to find a very stressful, touchy atmosphere if working in companies that go so far as to advocate the bullpen approach. By contrast, it is quite telling that many of the best (or at least most profitable) software companies, including Microsoft, provide developers with individual offices or at least cubicles.

Companies that truly need to use bullpens usually have systems that either have, or are heading toward "catastrophic coupling." Atlas Solutions, for instance, has a high availability ad serving system. On maintaining and developing this system, Scott Case said,

> Tradeoffs involve "keeping the lights on" versus innovation. We're dealing with 110,000 ads per second. 6 billion ads a day. Our SLA (software license agreement) says we must respond within 2 milliseconds...New features have a deeply cascading effect into the system, which is tricky considering you have to keep the thing up 24x7...Most of the five teams are in Seattle. We use a Bullpen style area and XP-style pair programming.

There are two possibilities here. Managers at Atlas Solutions would certainly argue that the bullpen approach is simply necessary for such a system. Speaking hypothetically, the system may actually seem to need the bullpen approach because of its tightly coupled, relatively inflexible architecture innately difficult to modify. Atlas's product architecture was not evaluated, but certainly there are many organically grown and tightly coupled commercial software architectures in the field.

Such systems could be said to have "tar-baby architectures." The more tightly coupled the architecture, the more developers have to be in one place to modify anything or to add features. Ironically, the more people are in one place, the greater the ability to modify the tar-baby architecture with even more esoteric, tightly coupled hacks: the tar baby grows even bigger and more developers become "stuck" to it (need to be in the same place to modify it effectively). Thus system size and complexity slowly increases. The bullpen grows. Individual developers have an increasingly more difficult time adding changes. The interview with Scott Case lends anecdotal evidence to this theory:

> There is one outsourced development team in India, but the software modules must be very well defined...the outsourcing hasn't worked out well. We seem to have difficulties making outsourcing work; it is very hard to carve off modular tasks.

Subjectively, software shops using bullpens are not particularly pleasant places to work, at least as a software engineer. Objectively, there is clear anecdotal evidence here (and there are many other examples) that bullpens may function as substitute—cheap in the short term and prodigiously expensive in the long term—for modular, well designed software architecture lending itself to clean division of labor. Bullpens may even facilitate an increase in architectural coupling and complexity over time: a detailed study worthy of its own thesis may be in order. Meanwhile, Product Managers should consider use of a pure bullpen development strategy to be a yellow flag: the ability to inject new features into systems developed this way may be quite high in the short term, but over the long haul, such shops expose themselves to high employee turnover and brittle, tightly coupled software which becomes increasingly difficult to modify or even outsource functionality enhancements. Here, there be strangler trees!

Skepticism on Agile Methods and Software Architecture

Agile methods are quite fashionable and popular in the software industry at the moment, and are certainly here to stay for the foreseeable future. Some important thinkers are adamantly opposed to this approach to software. MIT CSAIL professor Daniel Jackson had this to say on the subject (entire interview transcript in Appendix C):

> **PROFESSOR JACKSON:** You have to remember that the software industry, by and large, is extraordinarily conservative. Which is very surprising for a bunch of people who consider themselves radicals.
>
> **JOHN HEMPE:** Define conservative in this context.
>
> **PROFESSOR JACKSON:** They don't want to change the way they do things. They want to rely on, sort of, lift your finger and feel the wind. They don't want to consider the possibility that computers could help you

design software. You look, for example, at the astonishing vehemence with which Extreme Programming has attacked the idea of design.

JOHN HEMPE: Yes.

PROFESSOR JACKSON: It is not the method. It's the idea that you could design a system. It's deeply unfashionable to suggest the idea that a system could be designed.

JOHN HEMPE: True. These methods like Agile, Scrum, XP, these tight iterative methods--

PROFESSOR JACKSON: What they're confusing is the principle of risk management in which you don't want to make decisions for which you don't have information, versus the idea that even when you're capable of making predictions and good decisions you should just blunder ahead anyway and pay whatever price it costs to fix it.

What they essentially do is all the design in the code, which is an extraordinarily expensive way to do things. In the face of all their criticisms of formal methods, it's amazing how much work these people are prepared to do.

Nonetheless, legions of programmers work using these methods, perhaps because agile methods leverage their core skill: to write code. Myriad managers trying to keep some control over chaotic software projects love the methods because they at least allow quick response to ever-changing requirements. We will delve deeper into this topic in the section on Requirements Engineering.

Capability Maturity Model for Software

At the opposite end of the spectrum from Agile Methods is Capability Maturity Model-style project management. CMM is perhaps the most famous example of an attempt to organize all software management.

In November 1986, the Software Engineering Institute (SEI), with assistance from the MITRE Corporation, began developing a process maturity framework that would help organizations improve their software process...The CMM presents sets of recommended practices in a number of key process areas that have been shown to enhance software process capability. The CMM is based on knowledge acquired from software process assessments and extensive feedback from both industry and government. The Capability Maturity Model for Software provides software organizations with guidance on how to gain control of their processes for developing and maintaining software and how to evolve toward a culture of software engineering and management excellence.[22]

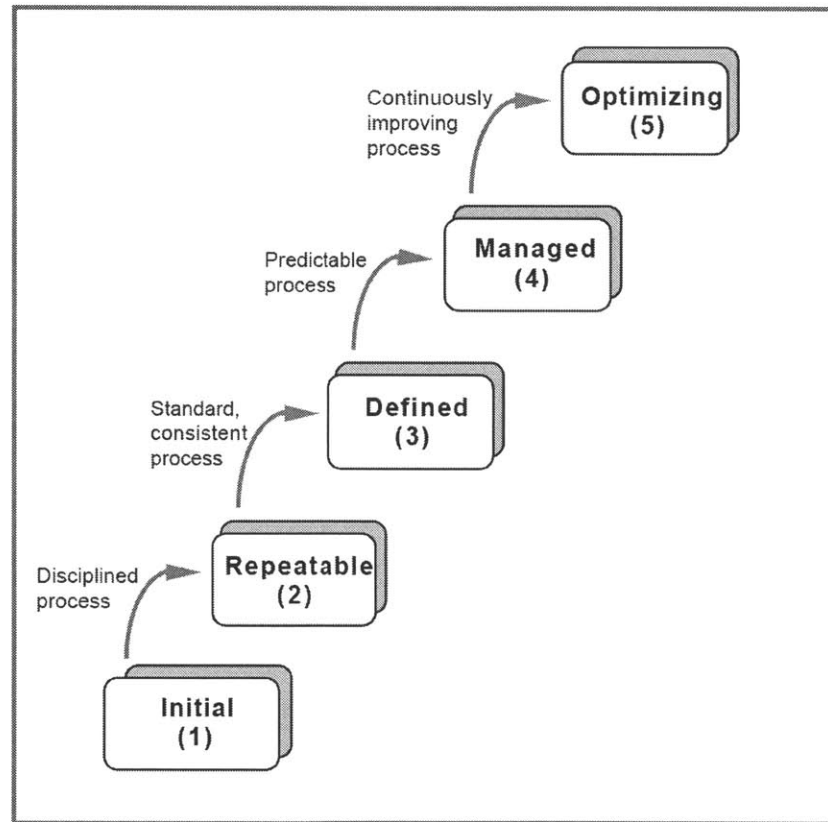


Figure 10 Levels of the Capability Maturity Model [22]

CMM focuses on the immaturity / maturity of a software organization's software development management. Five major levels of maturity are defined. Level 1 is total immaturity and Level 5 can take up to a decade to achieve. The defined levels include Initial, Repeatable, Defined, Managed, and Optimizing, as shown in Figure 10. The model asserts that these levels must be achieved in sequential order because each level of maturity uses the previous one as its foundation. Key features include commitment to performance, ability to perform, activities performed, measurement and analysis, and implementation verification.

The CMM model is noble enough in intention and certainly cannot be condemned as a method unilaterally. Many Indian IT outsourcing outfits, including Infosys, are certified CMM Level 5. A Product or Program Manager looking into outsourcing features of a product might want to take note of whether the company under consideration has CMM certification. Detractors of the model would note that an organization can take a shallow "test passing" approach to becoming certified at successive CMM levels, achieving high levels without becoming top quality software producers or service providers.

Software development is such an immature discipline that there is a great danger in "official" certifications of such models as CMM, which can still very much be considered developmental, if not experimental. A company can spend hundreds of thousands or millions of dollars being certified at a high level of CMM for marketing reasons, but ending up with a checkbox-filling, letter-of-the-law mentality that adds process for the sake of process, unnecessary layers of bureaucracy, and a net slowdown of software development. An advocate of CMM might argue that the model is so good

that a software organization is forced to be better if it's certified at a high CMM level even if it is just following the letter of the law.

The Immaturity of CMM

An article by Bach points out that CMM is funded by the US Department of Defense, and makes the assertion that CMM cannot "legitimately claim to be a natural or essential representation of software processes." He is concerned that CMM could be dangerous if too much focus is put on certification via CMM as a marketing tool, or as he puts it, "a whitewash that obscures the true dynamics of software engineering, suppress[ing] alternative models." [23] CMM is notably unpopular amongst highly nimble software companies such as those producing shrink-wrap software, and is unheard of in Silicon Valley Internet startups, unless perhaps they are outsourcing some feature development to a CMM-certified shop in India.

CMM itself claims not to be a "silver bullet" *à la* Fred Brooks, but with such heavy government funding its proponents tend to behave contrary to this humble posture. Process trumps people in CMM, according to Bach, but Cockburn and Highsmith assert in their Agile article that people actually trump process, and Bach is in their camp.[21] Software problems are simply too complicated to be codified as process: you can't make a human being a good chess player simply by giving him a good process, asserts Bach, and neither can you make him a good software developer.

The much tiered structure of CMM itself is problematic, according to Bach: the driving force is just to get to the next level. Defect prevention is primarily a Level 5 issue. This could possibly cause an organization obsessed with CMM certification for government contracts or general marketing to become blind to the need for careful defect

prevention regardless of the level of CMM maturity. Perhaps the most damning assertion of all Bach makes is that CMM stifles innovation through its slavish devotion to tiered process, through its ignoring of innovation or personal mastery as factors in the software organization. CMM is seductive to managers because it gives an easy formula to make some sort of process to the incredibly messy act of team software development. Blind management via CMM could steamroller blindly over personal creativity.

Michael Cusumano points out in his book, *Business of Software,* that Japanese companies have taken a zero-defect approach to software, but that this limits their commercial viability—although the approach makes for great embedded systems software in certain hardware devices.[15] CMM could degrade a software shop to the Japanese "software factory" level: it might work to slowly produce high quality embedded systems software, but could slow an organization down far too much to be viable in the commercial software market. At worst, a slavish devotion to CMM as a heavily funded, well known measuring stick useful as a marketing claim could gut out the innovative soul of a development organization and send the passionate and master coders running for new opportunities in less stifled shops. However, it at least advocates discipline and continuous improvement and is less likely to produce the "tar-baby architectures" and unpleasant working conditions (such as bullpen programming) of supposedly agile methods taken to extremes.

Process, Culture, and Project Management

Now that we've looked at both ends of the spectrum, extreme Agile methods and ultra-conservative CMM, let us pop up to a philosophical level. People, not project methods, ultimately do the work. In his article, *Enough About Process: What We Need*

are Heroes, Bach talks about the "three P's" for software managers: People, Problem, and Process. His basic argument is that process is peripheral rather than central to the development of excellent software. He concludes that the "heroic efforts of a dedicated team" are central to the success of a software project, which is certainly true. He believes that people accept roles more readily than tasks, and that people need to see that their tasks have meaning in terms of creating solutions that solve problems. [24]

People do accept roles more readily than tasks. The amount of structure placed on top of these roles in the form of formal software product management methodology is a key strategic decision based on individual company culture, as well as the product type under development. Department of Defense contractors may actually be forced to use CMM, while many Silicon Valley startups use no formal software project management process or "wing it" using a few Gantt charts generated using Microsoft Project. A lack of project management structure can lead to poor requirements organization and brittle product architecture, while heavy processes at large companies can degrade into slavish bureaucracy, slowing software project progress to the pace of molasses in November.

Product, Program, and Project Managers must realize that "software engineering" has a long way to go before it is truly reduced to an engineering discipline. Completely defined processes, such as those used in hardware manufacturing, simply do not work to create software products. Bill Gates has been quoted as saying, "A great lathe operator commands several times the wage of an average lathe operator, but a great writer of software code is worth 10,000 times the price of an average software writer." With such wide variability in individual productivity and the difficulty of predicting progress in the experimental and creative act of writing code, no single prepackaged process can meet a

company's project management needs. Processes must be tailored to company culture and to the product being developed. Continuous monitoring, as well as continual openness to process adaptation, are also vital keys to achieving project management success.

VI. Requirements Engineering

The Product Management Frontier

Now that we have explored Product and Program Management from the perspectives of history, economic environment, directly in terms of the jobs and roles, an extended case study, and the tightly intertwined cross functional role of project management, let us look to the future before we conclude. Where is Product and Program Management going from 2006 into the future? Requirements Engineering is a hot and controversial topic as of this writing. We will take a brief look at the current and most common methodology of requirements gathering, and then quickly advance to explore the frontier of Requirements Engineering. An advanced and controversial semi-formal methods technique from MIT's CSAIL department is reviewed, though such a method has limited commercial application in the short term. More commercially palatable methods taking into account socially complex business processes are then reviewed, including intent specifications, Quality Function Deployment as borrowed from automotive engineering, and finally, Product Roadmapping.

Why is Requirements Engineering such a hot topic today? Perhaps because Project Management has been studied intensely for many years in software engineering, yet still software projects are incredibly problematic:

> An astonishing *84 percent* of all software projects do not finish on time, on budget, and with all features installed, according to a survey by the Standish Group, which studied about 8,000 software projects in the United States in 1995. Furthermore, more than *30 percent* of all projects were cancelled before completion. The rest ran significantly over deadline and were 189 percent (on average) over budget. [25]

Obviously, Project or Development Management techniques alone are either not working very well or are not the core of the current problem. Traditionally, software requirements analysis was considered a relatively easy part of the development process. More recently, it has become increasingly recognized as being the most vital part of the process; given that the failure to identify requirements properly makes it virtually impossible for the finished piece of software to meet the needs of the client or to be finished on time. The term *Requirements Engineering* emerged in the early 1990's to stress the importance of treating requirements gathering as an analytical and not just a managerial process. The topic is gaining momentum and increasing interest in today's market, with companies becoming more aware of the magnitude of costly ripple effects from poor Requirements Engineering processes.

Current Requirements Engineering Practice

In most commercial companies, requirements are not so much "engineered" as gathered and tracked. At the simplest level, requested features must be tracked and prioritized to help capture requirements in an evolving system. Chapman notes,

> One of the chief tasks of a Product Manager is managing what is often called the wish or "tick" list—that very long list of features and capabilities requested/conceived of by your developers, customers, user groups, sales force, the CEO's spouse, etc. [11, p. 571]

Some companies try to over-prioritize bugs and feature requests. Do organizations really respond to five different levels of bug- and feature request tracking? Three is sufficient, is likely to cause less confusion and fewer items falling into grey areas of long term inaction. Chapman suggests the three categories for feature requests as *must haves, should haves,* and *nice to haves.* Given that many feature requests will

quickly pile up even for a medium-sized product once it hits the market, a company will need to establish a system of managing feature requests and their state in the development schedule. Chapman suggests minimum functionality for such a system should include the following abilities:

- Add feature requests to a central database or list
- Track when a request was made and total up multiple requests
- Map those requests back to a person or group of people
- Categorize feature requests
- Prioritize feature requests
- Keep track of when and whether a feature has been updated
- Tie a feature request to an external document such as a use case [11, p. 573]

Many companies, in the author's experience, maintain feature requests within a defect database or bug tracking system, with the designation "enhancement" to separate them out from bug fix requests. In an IT department not releasing software as a product, sometimes an entire release can be based on prioritized versions of these "enhancement bugs". It is to be hoped that an actual software product company would use more sophisticated mechanisms to stay in touch with the market, but this does not always happen. The author has certainly experienced, in startup companies, features being implemented on an instant, emergency mode basis because a single customer or the CEO wants a new feature. This is not universally a bad thing, but such crisis management as a habit does not make for stable software products providing added market value.

As an example of how rampant poor requirements engineering is in software companies, the number one answer Product Managers gave in response to complete the statement, "If I could say one thing to our company president without fear of reprisal, it would be…" was as follows: "get a long-term strategy that doesn't change with each

sales contract." [13] The author's experience correlates positively with this survey quote: the amount of Requirements Engineering executed based on sales contract crisis is astonishing.

Software Requirements: Never Understood from the Start?

Professor Daniel Jackson, one of the keynote speakers at the 13th International Conference for Requirements Engineering, interviewed for this thesis. His idea is that companies simply are not understanding software requirements from the start, and do not appreciate the technical nature of the problem. The following quotation has company-specific comments suppressed at his request (the whole interview transcript is in Appendix C):

> **PROFESSOR JACKSON:** Why think about requirements, because they change? Right? Now, I don't believe this. And the reason I don't believe this is the following...[although] requirements shifts are a huge source of complexity in system development, the XP people are working, by and large, for very conservative companies....My guess is that the reason the requirements changed is because they never bothered to really understand. I'm being very polemical now. But I think what often happens is, from the software engineer's perspective the requirements *have* changed. But the [business layman] on the other side says, "How did you ever not understand that? What did you think we were doing?" And they're amazed at the idea. [They're thinking], "We didn't change our requirements; we always wanted to do it like this." So, there's just a reluctance to spend time with the customer and truly understand what their problem is.
>
> **JOHN HEMPE:** Right, right. And that's definitely part of what I'm exploring. A typical quote from an average coder I know is, "I've never met a competent marketing person." It just seems like somewhere the pipeline of actual requirements to engineers who write code gets broken

down, and then the engineers get blamed for everything, or [perhaps I should say], the programmers get blamed for everything.

PROFESSOR JACKSON: Right, exactly. That's exactly what happens. And one of the reasons is that the initial attempt to characterize the problem is not sufficiently appreciated as a technical problem. It's thought to be something that needs less technical expertise. But it needs *more* technical expertise. It's the essence of the whole enterprise: figuring out what you're trying to do. So, that, I think, is a large part of the problem.

Another part of the problem is, Product Managers and other Marketing personnel with the business and "people skills" to connect with customers simply do not have the same level of technical expertise as the average programmer, let alone more expertise such as a software architect might have. There is an intangible, long, "lossy" pipeline between what software functionality "the market" actually will pay for and executable software code. Requirements flow into this pipeline from external sources such as customers, sales, prospective customers, competitive analysis, industry analysts, and consultants. Engineering, technical support, partners, and other employees can also serve as internal sources of requirements. Generally, the pipeline in a full scale company would be filtered and would flow through Product Managers, to Program Managers, to Software Architects and Developers. At this last stage, the Developers are, in an ideal world, presented with the highest priority features with maximum profit generating potential. They write the code to create these features, and the software sells like hotcakes to lumberjacks in an Alaska January.

Unfortunately, this pipeline has major flaws. Many of the most profitable requirements are never pushed into the pipeline as companies do a poor job of listening to the market. Requirements "leak out" and are lost due to poor communication between

stages. Requirements become distorted and mistranslated due to miscommunication between stages, much of this due to the generally increasing level of technical and decreasing level of business knowledge as the pipeline approaches the coding stage. Some *unprofitable* requirements may sneak into the pipeline at the last stage as Developers sneak pet features in with required features. Another version of this occurs when Developers expend excessive energy making a feature far more general-purpose than it needs to be out of the often justified fear that if they do not over-engineer the feature, Marketing will come back after seeing Feature X working and immediately ask for Feature X to do X + 1.

The Werewolf Theory of Requirements Engineering

So what do we do about this broken requirements pipeline? Let us look at the problem philosophically, and at one solution posed at MIT CSAIL, the Computer Science and Artificial Intelligence Laboratory. It is an accepted law of software development "physics," revealed to the community by Fred Brooks in his famous *No Silver Bullet* article, that "there is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity."[26] Requirements engineering is an intractable problem, a problem impossible to solve at all, let alone at a stroke with one management technique or well-defined process. Jackson is certainly correct that technical expertise helps, and perhaps the technical needs of RE are underestimated in today's companies. What Jackson may underestimate, though, is the sheer humanity and breadth of the pipeline involved. The MIT community tends to believe in heroic intellectual efforts as the norm, and the number of stakeholders is limited in a research context, but this is not

business reality. Numerous stakeholders with radically different self interests all act as lossy conduits of software requirements between the market and software implementation. Jackson admits it is difficult to stop and focus on requirements in the business world:

> **PROFESSOR JACKSON**: "We're going to lose a week's productivity by drinking coffee and chatting?" You've got to be a gutsy manager to do that, right? And I can understand that. From a research perspective, we do that all the time, because we don't have anyone breathing down our necks. We have much more flexibility in terms of when we deliver value, and so on.

The Silver Alloy Bullet

Jackson is an advocate of formal methods, and has a fascinating attempt to advance the science of the Requirements Engineering problem embodied in the invented language Alloy, product of research funded by the National Science Foundation, and by the High Dependability Computing Program from NASA Ames Research Center. Jackson claims Alloy is simpler than a programming language and is an excellent way of specifying the "schema" of a program.

> **PROFESSOR JACKSON:** Alloy, to a novice, might seem complicated, but semantically it is enormously simpler than any programming language. I mean, basically, all you've got is sets and relations, that's it. It's like a database schema but...actually; it's basically the same as a relational database schema.

Alloy was thus evaluated for suitability as a tool for Requirements Engineering by a technical Product Manager. (Important note: it was the thought of the author—not an assertion by Jackson—that Alloy might turn out to be a "silver bullet" for Requirements Engineering.) A seemingly well-written tutorial on Alloy did not quite correspond with

the functionality actually available on Windows XP. After spending several hours learning the initial semantics, such as the "3 levels of abstraction to understanding Alloy"—object oriented, set theoretic, and atomic/relational—the author spent several more hours attempting to understand and run the examples. The interface seemed friendly at first, but beyond the most trivial examples there were gross mismatches between the results pictured in the tutorial and the results shown in the actual program.

After a full working day on the problem, attempting to understand the complex set theoretic logic and to manipulate the graphical interface of the Alloy simulator to match the results in the tutorial, a level of frustration was reached as it was impossible to compare the results graphically. The author was never able to reproduce the graphical output, shown in Figure 11, for a simple file system. External help would be required, and the author followed an Alloy home page link to join a Yahoo group on the Alloy language. In the Yahoo group, the only activity within the past week had been a posting for a summer internship at the Jet Propulsion Laboratory, containing a link leading to a picture of a rocket scientist. Although hardly a *quod erat demonstratum,* this line of inquiry seemed to lend strong credibility to the author's suspicions, formed after many hours of mind-bending attempts to understand Alloy that formal modeling methods such as Alloy have not penetrated much below the level of the programmatic equivalent of rocket science in ease of use and accessibility.

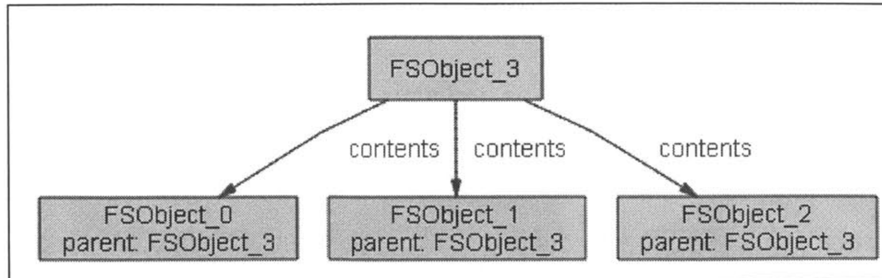


Figure 11 Graphical output of Alloy Modeling Language (Source: http://alloy.mit.edu/tutorial3/alloy-tutorial.html)

In fairness to the Alloy developers, they did respond to an email inquiry about the problems with the tutorial with a correction file for user interface representation. At this point, however, it had become clear that modeling in Alloy was a highly complex endeavor. Clearly, the language and methodology is not only far out of reach of the average Product Manager, but also not likely to be easily grasped by the average software professional in its present incarnation. Performing Requirements Engineering or system modeling in Alloy could easily become its own thesis topic, and may be an area for further work when the interface has been evolved.

Professor Jackson had claimed Alloy was "quite simple" as it was composed only of "sets and relations", but complexity is like broccoli that must be eaten: one can push it around on one's plate, but the total mass to be consumed will remain constant. The simplicity of only two fundamental concepts of "sets and relations" is unfortunately belied by the mind-bending and myriad ways one must manipulate them using pseudo-object oriented syntax that may seem simple on the surface, but is extremely difficult to master. Set theoretic thinking is far different from the object oriented procedural programming models in the minds of most programmers today.

The comparison to an RDBMS schema breaks down in that an RDBMS schema is based on tables, easily visualized as rows and columns, enforcing mentally clearer

limitations on the solution space than a set-theoretic modeling tool that can model any system. With advancements in ease of use, Alloy may one day prove a useful tool for technical personnel such as Software Architects, but it is unlikely to develop into something a Product Manager would use directly.

Bulletproof Social Processes

Formal methods of so-called "program proving" have been around a long time, but never gained much traction in industry. Alloy is part of the "lightweight formal methods" movement, but "lightweight" is perhaps a misnomer here. An adult blue whale on earth is lightweight compared to an ice cream scoop of matter from a black dwarf star. Analogously, "lightweight" formal methods are only light in comparison to true formal program verification, accepted by the computing community as virtually impossible for even modestly sized computer programs. Formal methods were falling out of favor as early as 1979, when a paper entitled *Social Processes and Proofs of Theorems and Programs* argued,

> Formal verifications of programs, no matter how obtained, will not play the same key role in the development of computer science and software engineering as proofs do in mathematics. Furthermore the absence of continuity, the inevitability of change, and the complexity of specification of significantly many real programs make the formal verification process difficult to justify and manage. [27]

Lightweight formal verification advocates believe that programming can be more like mathematics, and tools like Alloy exist in an attempt to show that, if programs themselves cannot be proven, at least system specifications can. Perhaps this is so, but such proofs do not get rid of the social issue:

> We believe that, in the end, it is a social process that determines whether mathematicians feel confident about a theorem—and we believe that, because no comparable social process can take place among program verifiers, program verification is bound to fail. We can't see how it's going to be able to affect anyone's confidence about programs. [27]

Jackson would surely argue vehemently against this argument applying to lightweight formal methods the same way it applies to formal verification and his research seeks to prove the contrary. It seems there is still a long way to go in making lightweight formal methods straightforward to use, and the fact is that industry has thus far largely eschewed lightweight formal methods as intractable. Jackson discussed the difficulties of getting industry to buy into formal methods in his interview:

> **JOHN HEMPE:** I think part of it is getting something well known and [professionally] leverageable enough.... It's a matter of showing the power, and showing that an average professional can think on that level. And I sometimes wonder if we sometimes just run into cognitive limits where it's really hard to think in these advanced ways.
>
> **PROFESSOR JACKSON:** But it's just abstraction, right?
>
> **JOHN HEMPE:** Absolutely. Abstraction is the most noble, but also the most difficult form of thinking.
>
> **PROFESSOR JACKSON:** I agree, I think you're right, but why should you think the average professional would need to do that? Why isn't it just like the schema designer?
>
> **JOHN HEMPE:** The Architect.
>
> **PROFESSOR JACKSON:** Yes, the architect. It's going to be a smaller group of people who do that kind of work.
>
> **JOHN HEMPE:** Right, right.
>
> **PROFESSOR JACKSON:** And the rest of the people will do much more concrete work, dealing with the details of the code.

JOHN HEMPE: So the important thing is to have a few key experts who are able to specify the system in this formal language.

PROFESSOR JACKSON: I think that is essential.

JOHN HEMPE: I agree, in principle.

In practice, however, each and every programmer at most software companies acts as a mini-architect when coding individual features. This is certainly true at Microsoft. They might stay within generally framed architectural boundaries, but certainly not within anything as specific as a program schema like Alloy or any other lightweight formal method tool would produce.

A small group of architects can specify a skyscraper or a bridge and then farm out the work to construction crews and engineers, but software development is quirky, more creative and artistic than a "mere matter of engineering" in most cases. The side effect is that software development simply does not have such clean division of labor between architects and engineers. Perhaps the division of labor problem, the difficulty in having a bright, clear line between the few architects and the many engineers, partially explains industry resistance to the kind of hierarchical engineering organization lightweight formal methods imply.

Capturing Requirements with Social Process Methods

Formal and semi-formal methods do not seem to solve the software requirements pipeline problem, at least for average commercial projects. Sheer complexity and a lack of advanced tools make them inaccessible to average professionals. They do not take into account the "dirty" problem of social and business complexity, although advocates might argue this is not their job. Regardless, methods are needed that do help with the

social process, rather than focusing completely on verification and modeling. We will look at three such advance methods, roughly in order of increasing practicality and current usage. The methods are *Intent Specifications*, *Quality Function Deployment*, and *Roadmapping.* The first method is currently in use mostly in the military/aerospace industry; the second is in limited use in the software industry, although several automated software packages have been written to facilitate its use for the purpose; and the third is in fairly widespread use by software companies both as a Requirements Engineering tool and as a preemptive marketing tool.

Intent Specifications

Intent Specifications is a method proposed by MIT Professor Nancy Leveson to facilitate "grounding specification design on psychological principles of how humans use specifications to solve problems as well as on basic system engineering principles."[28] Taking a cognitive psychological approach, she notes that interface design must address the aspects of *content, structure,* and *form* to serve as an effective medium. A Product Manager would not necessarily be involved in all levels of creating this kind of specification, but the interesting thing about the method is that it is "rooted in abstractions based on intent." Intent abstraction is shown on the vertical axis in Figure 12.

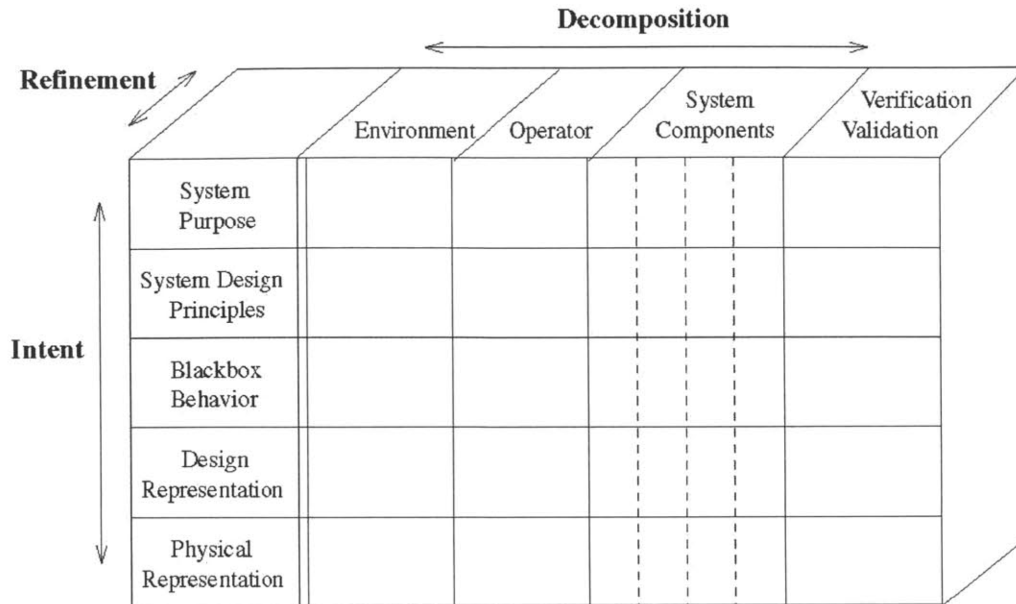


Figure 12 Intent specification structure for software systems [28]

A detailed study of the method is beyond our scope, but the advantages of capturing and systemizing *intent* rather than only *functionality* should be clear to the Product Manager. Ideally, the PM discovers and specifies the market's intent, or goals, for a system (or an upgrade) it will purchase, while it is in the realm of engineering how the system realizes the goals with specific functionality. Leveson's driving example is TCAS II, the Traffic Collision Avoidance System for aircraft. As an example of top level system intent, the *System Purpose (goal),* with G1 being the high level goal and R1 as a refined subgoal, would appear as follows:

> G1: *Provide affordable and compatible collision avoidance system options for a broad spectrum of National Airspace System users.*
> R1: *Provide collision avoidance protection for any two aircraft closing horizontally at any rate up to 1200 knots and vertically up to 10,000 feet per minute. [28, p. 10]*

The PM would be primarily responsible for the first two levels including the *System Design Principles.* At this level, engineering consultations begin as this level

specifies the principles needed to realize the top level purpose behavior. In the TCAS II example, one example of a principle is this:

> PR1: *Each TCAS-equipped aircraft is surrounded by a protected volume of airspace. The boundaries of this volume are shaped by the tau and DMOD criteria.* [28, p. 14]

The method of Intent specifications works by deriving the system through successive layers of human intent-based abstraction, thus simplifying requirements from the often huge laundry list of functional specifics. Noted computer scientist and software reliability specialist Martyn Thomas, founder of Praxis software (and a believer in lightweight formal methods) recently wrote,

> Requirements are often complex because they contain unnecessary implementation detail, or elaboration of many special cases that could be better expressed as a general principle. [29]

Simplification of requirements expression is thus one major advantage of Intent Specifications. Another advantage is that system intent is often lost in large laundry lists of desired functionality; this method makes intent explicit by centering the specification process on the cognitive psychology of human intent. This seems like a good idea, given that people, not functionality list processing machines, will write and use the resultant software.

QFD: Quality Function Deployment

The QFD design tool originally had nothing to do with software. According to Hauser and Clausing in their much-cited Harvard Business Review article, *The House of Quality,* QFD "originated in 1972 at Mitsubishi's Kobe shipyard site. Toyota and its suppliers (such as Bridgestone Tire) then developed it in numerous ways." [30]

Eventually, American auto manufacturers and other manufacturing companies began adopting the methodology, which Hauser and Clausing call "a kind of conceptual map that provides the means for interfunctional planning and communications."

With roots deeply in manufacturing, the House of Quality was first academically applied to software applications in a 1990 MIT thesis by Laura Donohue entitled *Software Product Development: An Application of the Integration of R&D and Marketing via Quality Function Deployment*, exploring "the *integration of corporate functions* to provide the cross-fertilization of functional technologies and ideas, as well as the communication of independent functional needs during the process of product delivery."[31] Today, while QFD hardly dominates the software world, it has become an important part of the "Six Sigma for Software" movement. Several QFD / House of Quality software products intended for software design exist on the market today, and an extensive Web community exists in support of the topic, including such sites as the following: http://software.isixsigma.com/, a site supporting QFD and other Six Sigma techniques adapted for software.

The positive side of the QFD technique for software is the integration of marketing (customer) and R&D perspectives. A schematic version of the House of Quality is shown in Figure 13. Customer needs are listed along the vertical axis (rows) *Customer Requirements,* and software features are listed along the horizontal axis (columns), labeled *Supplier Measurable Responses*.

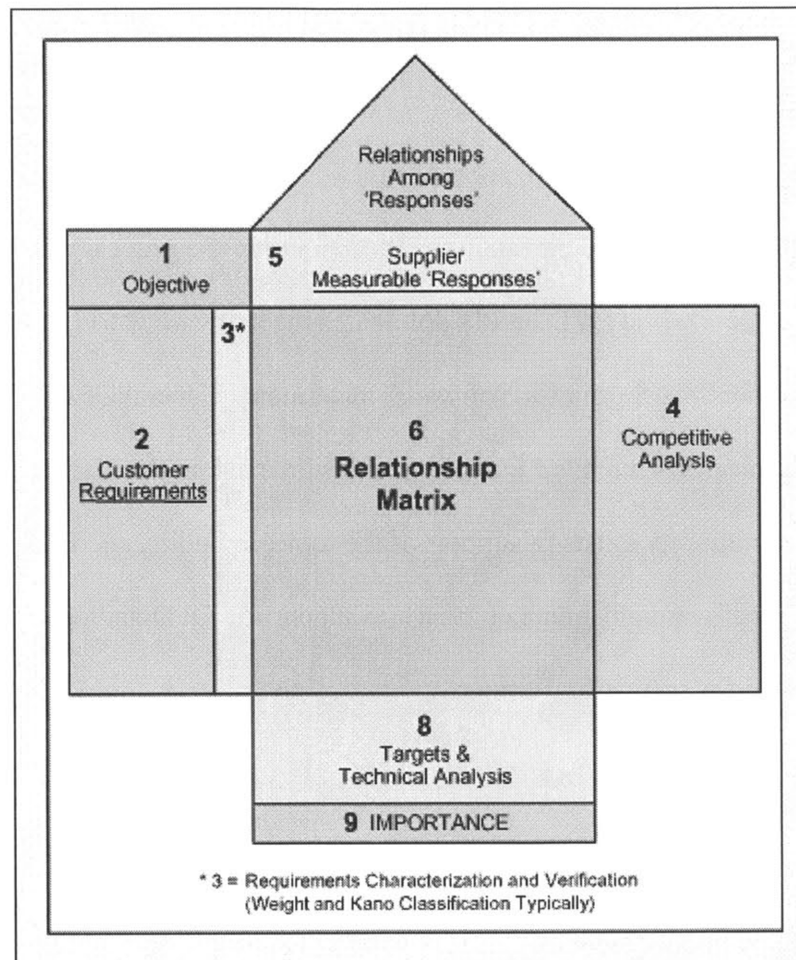


Figure 13 Basic QFD House of Quality (Source: http://software.isixsigma.com)

Although a useful tool, the QFD technique has some weaknesses when applied to software. An article on the Six Sigma for Software Web site notes the translation issues in applying a manufacturing model to software:

> Software design presents interesting challenges for several reasons. Software is an intangible product that is not always conducive to explicit acceptance measures. Design elements are coupled and interdependent, which is different from physical designs that can be deconstructed into independent but functional sub-assemblies, parts, and components. Software is not so easily divisible, creating additional design challenges. [32]

The article goes on to suggest methods of modifying QFD for software, beginning with a definition of "relevant and tangible customer needs." The article references the ISO 9126 standard, structuring customer needs into six product characteristics: Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability. A long matrix attempting to build a taxonomy of customer needs based on these six attributes is given.

Software, being design embodied, tends to resist such general purpose taxonomies. QFD will not provide us with an easy, cookie-cutter approach to solve the Requirements Engineering problem, with or without laborious taxonomies and other attempts at adaptation from the manufacturing sector. It does, however, provide a good starting point for organizing and visualizing the interrelationships between perceived needs of customers and marketers versus developed and developing software features.

Roadmapping

Roadmapping is a forward-looking Requirements Engineering process. Whereas the other methods discussed involve capturing needs mostly for the version of a product immediately in the development pipeline, roadmapping often involves projections up to several versions or even years into the future. Richard Albright of the Product Development and Management Association defines roadmapping as follows: "Product-

Technology Roadmaps link market and competitive strategy to product plans to technology strategy – with quantitative targets and plans for achieving objectives."[33]

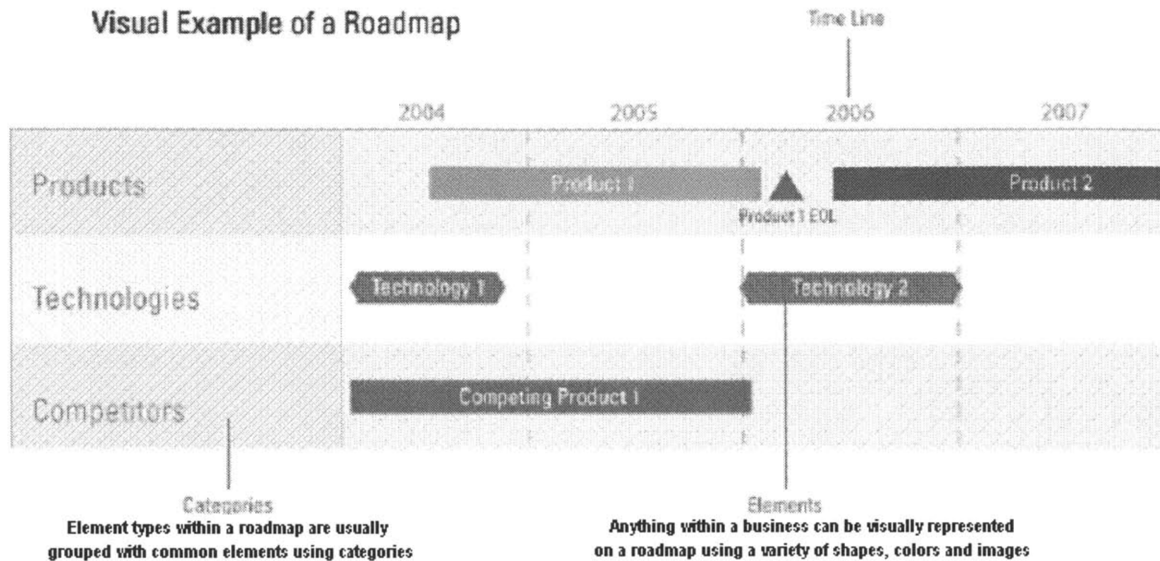


Figure 14 Visual Roadmap Example [34]

Although many consulting firms and software packages exist to support roadmapping, the process is neither standardized, nor has it been reduced to "mere engineering." Some companies use roadmapping for internal planning, while others use it more as a marketing tool. Alcatel performed a detailed study covering 246 projects on Requirements Engineering focused on reducing downstream requirements changes, noting, "A common denominator of requirements changes is that they practically always correlate with project delays." Changes occur for several reasons, the most important among them being, "stakeholders often do not agree on content and later demand changes," and also, "requirements are not uniformly visible and thus not agreed by different teams." Alcatel claims they were able to reduce delays over 20% per year using a variety of techniques, primarily by an increased "integration of upstream processes with the product life-cycle related RE processes," a form of technology roadmapping. Non-

roadmapping techniques were also used, such as, "install an effective core team for each product release," so the benefits cannot be conclusively or entirely attributed to roadmapping. [3]

Roadmapping as a Marketing Tool

Some professionals are more skeptical about roadmapping as a purely technical exercise. Chapman claims that product roadmaps are more marketing instruments than pure technical planning tools. He introduces roadmapping as follows:

> First developed in the 60s and 70s primarily as weapons in technology companies' FUD [Fear, Uncertainty, and Doubt] arsenals, product roadmaps are ostensibly sneak peeks into the future that also function as obstacles to competition. [11]

Microsoft, for instance, always seems to have a roadmapped solution to any software problem if they are not selling something in the space presently. Some of the few books on Software Product Management do not mention roadmapping at all, but books and Internet sources that do mention it place it in the realm of the Product Manager to create and maintain as a tool for coordinating stakeholders as well as for visualizing future strategy. Chapman gives a brief three-element list of the product roadmap components:

- Projected release dates for future versions of a product or product line
- A very high-level overview of new functionality
- A discussion of what new hardware and software platforms your software will be running on

He cautions, however, about revealing competitive information in a product roadmap, as roadmaps given to customers, or certainly to the press, will invariably fall into the hands of the competition. "If a roadmap is too feature-specific," he notes,

"you've just given your competition valuable information about future versions of your product well in advance of your product's release." On the plus side, roadmaps can be powerful instruments for maintaining current customers:

> One advantage of showing your customer a broad roadmap is that it can increase their confidence that the product they purchase will be supported and available in the future. In addition, showing customers a carefully crafted roadmap can support your sales of maintenance and service agreements. [11]

Benefits and Challenges of Roadmapping as an Evolving Method

Back on the technical side, one paper from the 13th International Conference on Requirements Engineering (2005) describes a University of Helsinki study of roadmapping as a way to "link the business view to requirements engineering." We will conclude our discussion of roadmapping with a list summarizing some of their most important findings:

- Roadmapping strengthened the link between business decisions and Requirements Engineering
- Product Managers saw roadmaps as tools for communicating their ideas to other stakeholders
- Developer viewpoint was less emphasized compared to that of other stakeholders (management, sales, channel partners, and customers)
- Practitioners complained roadmaps got immediately out-of-date
- Practitioners were missing ways to tie product development resources to roadmaps [35]

Engineering Software Requirements in the Future

All parts of software creation are rife with difficulty, as we are reminded by Brooks. Writing correct code is still more art than science and maintaining code is

extremely costly. Yet executing a process of Requirements Engineering to meet market needs—addressing individual customer needs as well as collective market needs without evolving a software system into a brittle and unprofitable corner—may be the most challenging problem in software product creation. Certainly, it is the most expensive problem if done poorly, as all phases of the product life cycle are affected by mistakes, omissions, and unnecessary additions at this early but critical phase. Semi-formal methods may not be the Requirements Engineering solution for commercial software, but Jackson was correct in saying, "It's the essence of the whole enterprise: figuring out what you're trying to do."

We have seen that today, many companies still succumb to a crisis-management approach to Requirements Engineering, or at best, keep a fairly informal prioritized list. With excellent Product and Program Managers, the prioritized list or database approach can and does work for companies. Methods do exist, however, to help managers and businesses to *engineer* their requirements rather than only to gather and best-guess prioritize them. At the very least, methods such as the matrix-based Quality Function Deployment may help managers to organize their thoughts and avoid blind spots in connecting product and technology requirements to profit generating market needs. Injecting a crisis-motivated feature now and then will not necessarily kill a product line, but is a bad habit into which companies can, and do, easily fall. A disciplined Requirements Engineering process can help a company avoid this habit by forcing a coherent vision for its product lines and their evolving feature sets.

VII. Conclusion

Product Managers face a challenging world in which emergent technologies and economics change perhaps faster than in any other product market in history. The original premise of the author going into this thesis considered Product Management to be primarily an engineering function. Research showed, however, that the function emerged from Proctor & Gamble's marketing role and bridges across to development from marketing rather than the other way around. The truly successful PM bridges the gaps in the corporation, connecting the points of Ebert's "Bermuda Triangle" between marketing, strategy, and engineering to prevent product value in the marketplace from mysteriously vanishing.

We have seen that Microsoft worked to bridge the marketing / engineering gap from both ends, hoping to meet in the middle, by creating an engineering-oriented version of the Product Manager in the form of the Program Manager. Microsoft's market success is undisputed, and this structure has been imitated by many successful software companies, but not all. Google, for instance, does not use the Program management role and instead splits the PM role into Business and Technical versions of the job. Half of all PMs are now involved in the writing of detailed specifications, and the technical demands of the job have recently been trending upwards.

On that front, the emerging field of Requirements Engineering was explored, but obvious, simple, or one-size-fits-all technical or managerial solutions to the problem prove elusive. As with many aspects of software product creation, the "ancient" wisdom of Fred Brooks' *No Silver Bullet* applies in this area. RE fails to yield to any single technical solution. Semi-formal models were explored, most notably the set theoretic

modeling language Alloy, but this method quickly proved worthy of a Computer Science thesis in and of itself. Further work in this area may involve exploration of ways to make Alloy usable for technical PMs. Also, domain-specific Alloy with pre-designed patterns for certain domains might make it more practically useful. The success of analogous specification languages such as SQL for RDBMS schema creation may be partially attributed to domain specificity.

Regardless of advances in Alloy or similar approaches, RE will prove intractable to solution by any "killer application" or tool. Enterprise software provides a perfect illustration as to why this is the case: such software models how people and organizations do business with each other. In such systems, all the intractable complexities of personal and organizational interaction are innately embedded. Problems arise from straightforward communication difficulties and issues such as the tribal nature of human beings. Chapman notes there is a "natural tension that exists between functional groups. Each group has its own perception of the value it brings to the group and stereotypical perceptions about the other groups....These tensions and issues are natural and cannot be reversed." [11, p. 567]

The PM's job will thus never be radically simplified by automated requirements capture tools. Inter-group tensions will always have to be managed, and as interviewee Philip DesAutels noted, alignment of stakeholders is a core competency of the successful PM. Marketers want to create collateral and buzz, salesmen want to sell, developers want to code, but it is up to the PM to hold up a hand and ensure that what Professor Jackson recommends takes place: "At the beginning, you should figure out what problem you're trying to solve, and you should have some fairly precise characterization of that

problem." Whether this is performed using semi-formal models for a high availability or high security product, or QFD, or simply keeping a well managed database of features, is beside the point. The PM and the systems architects must synthesize the vision for and maintain the conceptual integrity of the product.

Leadership, an easy word to use but a difficult one to define precisely, is ultimately the critical skill of the PM. Advancements in project management methods and Requirements Engineering tools may make the job incrementally more tractable, but no technical solution will obviate the need for PMs to be business leaders of cross functional teams. No single piece of advice can make a great leader. Interviewee Shuman Ghosemajumder, a top PM practitioner at industry leader Google, made this unintentionally sage comment about leadership:

> One thing I heard about Product Management is to remember as a Product Manager that you are the one person on the team that is completely disposable. If you always remember that, keep your eye on the ball of adding value to the team, then you will be successful.

Of course, the apparent paradox is that only through such humility does one have a chance to emerge as the successful CEO of the product, the "completely critical" person on the team. Ghosemajumder's comment dovetails nicely with Professor Giovanni Gavetti's closing comments on leadership in Harvard Business School's Strategic Reasoning Laboratory course: "The right combination of humility and thinking big is, I think, the key to being a great leader."

Excellent PMs are extremely valuable in the marketplace, because the ideal PM has a rare combination of people and technical skills. Good PMs are competent systems architects, but the process of gaining these skills is solitary and often limits the architect's

finesse with people. Those from a pure marketing background may have the finesse with people but lack systems architecture skills, damaging the conceptual integrity of the product vision. The best PMs have both of these opposing skills and are rare. As software product systems continue to increase in functionality and complexity, the value of this rare skill set will continue to increase, as will the need for better education in cross functional skill sets such as MIT's System Design and Management program is designed to provide.

Ultimately, it is impossible to create an instruction manual or otherwise easily capture all the elements of successful Product Management, just as there has never been a successful manual on how to be the CEO of a corporation. At the same time, no amount of process management can circumvent the need for the Product Manager. Philip DesAutels of Microsoft said in his interview,

> No matter what the process is, there's a little box where it says, "a bit of magic happens here." At the end of a car assembly line, there's a group of people called the "fixers" who do a lot of ad hock fixes. They might have hammers, two by fours, even strange things like pillows. A big part of Product Management is the "fixer" job, where the magic happens.

Attempting to subdivide the job further to avoid the need for interdisciplinary skills simply does not work, as this complex confluence of skills is precisely what is needed to perform the magic in that little box at the nexus between marketing, engineering, and strategic management.

Bibliography

[1] Eyewitness to History, *California Gold Rush*, http://www.eyewitnesstohistory.com

[2] Standish Group International Inc., *CHAOS Chronicles v3.0*, http://www.standishgroup.com/chaos/toc.php

[3] C. Ebert, *Requirements BEFORE the Requirements: Understanding the Upstream Impact, Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, 2005, pp. 117-24.

[4] CareerOverview.com, *Product Management Careers, Jobs, and Training Information*, http://www.careeroverview.com/product-management-careers.html

[5] Steve Johnson, *Role of Product Management*, http://www.softwareceo.com/white_papers/ProdMgrRole.pdf

[6] Hal R. Varian, Joseph Farrell and Carl Shapiro, *The economics of information technology : an introduction*, Cambridge University Press, Cambridge ; New York, 2004.

[7] Glenn Ellison, Sara Fisher Ellison and National Bureau of Economic Research., *Search, obfuscation, and price elasticities on the Internet*, National Bureau of Economic Research, Cambridge, Mass., 2004.

[8] Wikipedia, *Proctor & Gamble*, http://en.wikipedia.org/wiki/Procter_&_Gamble

[9] Michael A. Cusumano and Richard W. Selby, *Microsoft secrets : how the world's most powerful software company creates technology, shapes markets, and manages people*, Free Press, New York, 1995.

[10] Alyssa S. Dver, *Software product management essentials : a practical guide for small and mid-sized companies*, Anclote Press, Tampa, Fla., 2003.

[11] Merrill R. Chapman, *The Product Marketing Handbook for Software, 4th Edition*, Aegis, [S.l.], 2003.

[12] Daniel Condon, *Software Product Management: Managing Software Development From Idea to Product to Marketing to Sales*, Aspatore Books, Boston, 2002.

[13] Steve Johnson, *Annual Product Management Salary Survey for 2005*, http://www.pragmaticmarketing.com/productmarketing/survey/2005/index.asp

[14] Microsoft, *Technical Program Manager*, http://members.microsoft.com/careers/careerpath/technical/programmanagement.mspx

[15] Michael A. Cusumano, *The business of software : what every manager, programmer, and entrepreneur must know to thrive and survive in good times and bad*, Free Press, New York, 2004.

[16] Michael A. Cusumano and David B. Yoffie, *Competing on Internet time : lessons from Netscape and its battle with Microsoft*, Simon & Schuster, New York, NY, 2000.

[17] Robert P. Carroll and Stephen Prickett, *The Bible : Authorized King James Version*, Oxford University Press, Oxford ; New York, 1998, pp. Proverbs 16:18.

[18] George Lakoff, *Don't think of an elephant! : know your values and frame the debate : the essential guide for progressives*, Chelsea Green Pub. Co., White River Junction, Vt., 2004.

[19] Jim Highsmith and Alistair Cockburn, *Agile Software Development: The Business of Innovation*, Computer, 34 (2001), pp. 120-22.

[20] Robert C. Martin, *Extreme Programming Development through Dialog*, IEEE Software (2000), pp. 12,13.

[21] Jim Highsmith and Alistair Cockburn, *Agile Software Development: The People Factor*, Computer, 34 (2001), pp. 131-33.

[22] Curtis Paulk M, B., Chrissis, M.-B., Weber C. , *The Capability Maturity Model for Software*, Software Engineering Institute, 1993, pp. 26.

[23] James Bach, *The Immaturity of CMM*, American Programmer (1994).

[24] James Bach, *Enough About Process: What We Need are Heroes*, IEEE Software, 12 (1995), pp. 96-98.

[25] Detlev J. Hoch, *Secrets of software success : management insights from 100 software firms around the world*, Harvard Business School Press, Boston, Mass., 2000.

[26] Frederick P. Brooks, *No Silver Bullet: Essence and Accidents of Software Engineering*, Computer, 20 (1987), pp. 10-19.

[27] A. DeMillo Richard, J. Lipton Richard and J. Perlis Alan, *Social processes and proofs of theorems and programs, Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, ACM Press, Los Angeles, California, 1977.

[28] Nancy G. Leveson, *Intent Specifications: An Approach to Building Human-Centered Specifications*, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 26 (2000), pp. 21.

[29] Martyn Thomas, *A View from the Stern*, Safety Critical Systems Club newsletter; The Centre for Software Reliability at Newcastle University (2005).

[30] John R. and Clausing Hauser, D. , *The House of Quality*, Harvard Business Review, 3 (1988), pp. 63-73.

[31] Laura E. Donohue, *Software Product Development--an Application of the Integration of R&D and Marketing via Quality Function Deployment Sloan School of Management*, Massachusetts Institute of Technology, Cambridge, 1990, pp. 154.

[32] Dan Zrymiak, *Software Quality Function Deployment: Modifying the "House Of Quality" for Software*, http://software.isixsigma.com/library/content/c030709a.asp

[33] Richard E. Albright, *Roadmaps and Roadmapping*, http://www.albrightstrategy.com/roadmap.html

[34] Alignent Software, *What is Roadmapping?*, http://www.alignent.com/resources/articles/whatisroadmapping.htm

[35] L. Lehtola, M. Kauppinen and S. Kujala, *Linking the Business View to Requirements Engineering: Long-Term Product Planning by Roadmapping, Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, 2005, pp. 439-46.

Appendix A: The Great Windows / UNIX Platform Battle and Software Commoditization

Parallel Timelines in Hardware and Software

Everyone knows that Microsoft is ascendant in the world of prepackaged software products, and the Windows operating system dominant on Intel-based PCs. Most people know at least of the existence of UNIX, which seemed to be losing relevance for awhile until the advent of Linux and the new vitality of Open Source. Not everyone, especially younger IT professionals, understands how this came to be. The fact that there are specific parallel timelines in the development of the software product market may not be so obvious.

Why is this relevant to software Product Management? Both the markets for software products, and the stage of the competitive landscape, have been set by these powerful, parallel trends in software & hardware history. The future stage and market, furthermore, will be to some degree determined by the trajectory of these continuing timelines. The timelines we consider begin at the end of the Punch Card era and the beginning of the timeshared "green screen" terminal era.

The Mainframe to UNIX to Open Source Timeline

A picture is worth a thousand words, so we will begin the discussion with a graphical timeline covering the Mainframe hardware and Unix OS path. Of course, many, many things happened in computing during this period, only a few highlights are given for the purposes of illuminating this discussion.

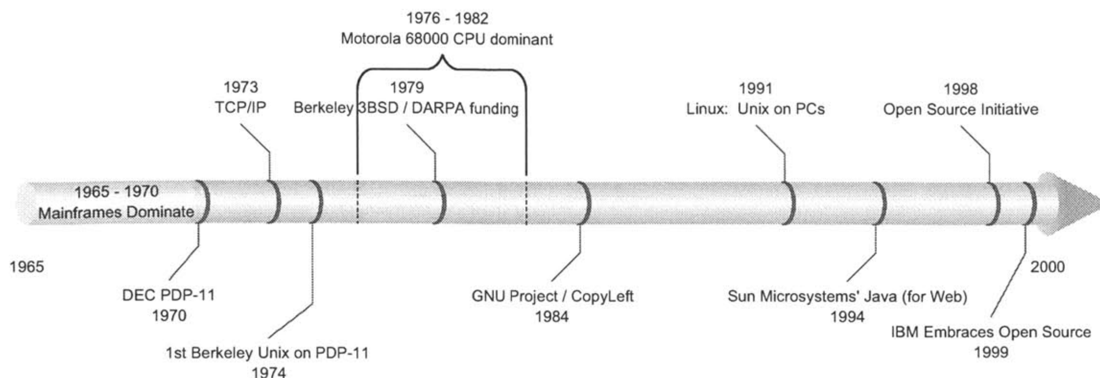


Figure 15 Timeline I: The Mainframe / UNIX Path

Early in "post-Punch Card" software history, mainframes (mostly IBM but also others, such as Burroughs, Sperry-Univac, and NCR) dominated business use. As 1970 approached, minicomputers arose, famously the PDP-11. The PDP-11 and VAX-11 machines, the latter being more business-oriented, became the reference systems for a new operating system called UNIX, which came out of AT&T Bell Labs. UNIX was not very important until the mid-1970's, when UC Berkeley's Computer Systems Research Group got ahold of it, the project being underwritten by the Defense Advanced Projects Agency (DARPA).

This project caused some landmark additions to UNIX and made it truly an industrial strength operating system: these additions included virtual memory, a robust and faster file system, and the vital TCP/IP network protocol in 1973-74. With the help of continued DARPA support, Berkeley continued to work on UNIX into the late 1970's, releasing the landmark 3BSD (Berkeley Source Distribution) in 1979. A lot of proprietary work was going on as well—this is nowhere near a full recounting of the history of Unix, as we are interested in the trend leading to open source—but the BSDs were instrumental in getting an entire culture of computer programmers used to the idea of sharing source code, mostly through creating and releasing utilities for the operating system along with source code.

Unix, especially the BSDs, along with the advent of fast CPU's, specifically the Motorola 68000 series during the mid-1970's to early 1980's, ushered in an era of ubiquitous and collaborative computing. Email and USENET groups (this was before they became drowned with Spam) helped advance this trend. The hardware was a

necessary backdrop, but it was UNIX that created a common, standard platform to allow the code sharing trend to achieve a critical mass. [12]

One early culmination of this trend toward ubiquitous and collaborative computing was the GNU project, started in 1984 by a group centered on Richard Stallman, who quit his job at MIT and created the immensely popular EMACS editor, further fomenting the subculture of shared code and eventually drafting the GNU General Public License or "copyleft". This constituted the first legally binding construct supporting the open computing philosophy. At this point, the foundations for open source had been laid. The term "open source" did not exist yet and would not be coined until about in 1998, but let us jump the track here in about 1984-85 and ride the parallel track of highly proprietary computing.

The Intel to IBM / Microsoft to Wintel Timeline

Many works exist to describe this parallel timeline, which created a business monster of proprietary computing, the richest man in the world, and such a powerful hardware/software duopoly as to incite numerous antitrust actions by the Department of Justice. *Microsoft Secrets* by Cusumano and Selby gives a detailed account of the Microsoft side and *Inside Intel: Andrew Grove and the Rise of the World's Most Powerful Chip Company* by Jackson covers the other. The reader is most likely familiar with this famous business story, so a sparse timeline and recounting should suffice.

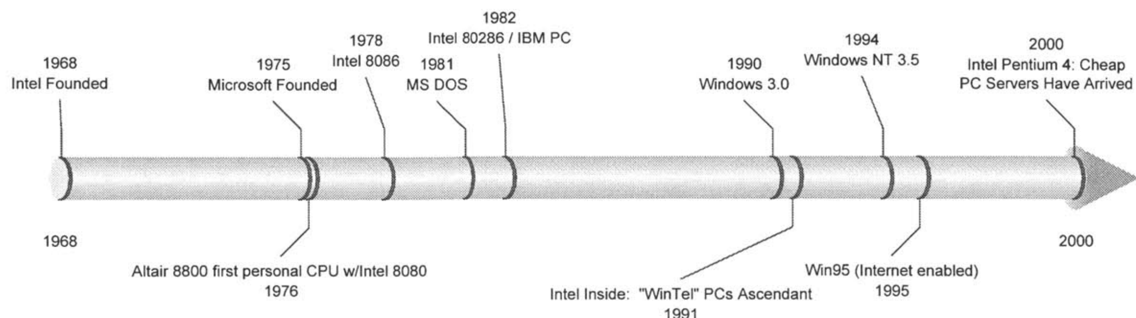


Figure 16 Timeline II: The Wintel / Proprietary Computing Timeline

This timeline starts about three years later than the previous one, in 1968 with the founding of Intel, and is essentially "asleep" until 1975-76, when Microsoft was founded and the Altair 8800 personal computer was released. Bill Gates, acting as an early "anti-Stallman", takes a passionately proprietary view of his programming work. The timelines nearly converged early on as Microsoft licensed UNIX and began working on XENIX, a PC version. However, this effort was eclipsed by the release of what became the prodigiously popular MS-DOS (Microsoft Disk Operating System) for IBM in 1981. Microsoft retained ownership of virtually all rights to the software, and in 1982, the ball was rolling with IBM PCs (or simply PCs) rolling out with the new Intel 80286 microprocessor running MS-DOS.

For our purposes, we can skip ahead to 1990 when the "Wintel" (Windows operating system running on Intel-based IBM clones) architecture computer truly began with Microsoft's release of Windows 3.0. Intel launched the historically successful branding maneuver represented by the "Intel Inside" marketing campaign in 1991. Embarrassing bugs and instability were largely corrected by the Windows 3.1 upgrade in 1992, and from that point on the Wintel architecture dominated the personal computer world, marginalizing all other players, most notably Apple computer.

Microsoft was fantastically successful on this timeline and became, perhaps, the most profitable business in history because they had the best of all possible worlds: they benefited from, on the one hand, an open hardware platform which created great economies of scale and choice in hardware. With a binary-compatible architecture and standardized hardware platform, thanks to reference designs created by the tight Intel/Microsoft relationship, the two achieved their own version of ubiquitous computing. Although Intel profited greatly, Microsoft was the truly historical winner because, as hardware became commoditized, they kept a fiercely proprietary hold on the software: the Windows operating system. This monopolistic stranglehold on the operating system acted as a gateway to allow them to become easily the number one seller of applications, most notably the set which became the Microsoft Office suite. The company was, is, and for the near future will remain the number one success story in Software Product Management, marketing, and sales.

Some attribute this to Bill Gates' brilliance and vision, others to his avarice and a lucky spot in hardware / software history. Perhaps, to some degree, both are true. Regardless, the Wintel architecture was ascendant by the mid-1990's, and slowly moved up-market as hardware advanced and the Windows NT operating system came into play. As 2000 approached, Intel Pentium processors arrived on the scene, cranking up toward the Pentium IV and encroaching upon the performance capabilities of industrial strength UNIX servers. At the same time, an ex-Digital Equipment engineer named Dave Cutler created an industrial strength version of the Windows operating system, the aforementioned Windows NT, with a Unix-like kernel.

The Non-Euclidean Software Market: Do Parallel Lines Ever Meet?

The parallel timelines were beginning to converge as the millennium approached, or perhaps "collide" would be a more appropriate term. The up-market migration of PCs with better operating systems and faster hardware began to threaten the idea of the "industrial strength" workstation and server, since PCs equipped with Windows NT 3.5 began to be good enough to act as servers.

We left the Mainframe – UNIX timeline in 1984-85. Another trend crossing over to threaten the workstation and server market on that timeline was the creation of Linux by a Finnish university student, Linus Torvalds. The most important thing Linux provided was the ability to run UNIX on the increasingly powerful Intel-architecture PCs. This further threatened the dominance of old school "industrial strength" workstations and servers such as those provided by Sun Microsystems. The UNIX world is thus being badly threatened by the PC world's economies of scale. Our timeline ends in 2000, but as of this writing in 2006, Sun Microsystems has never recovered at least in terms of stock price, and some believe they are languishing as a marginalized player due largely to the ascendance of Linux and server-powerful PCs.

So far, the threats discussed are all against the Mainframe / UNIX / high end server and workstation market, but the collision goes both ways. Intel architecture computers running Linux are notably not running Microsoft Windows. The Wintel

architecture started ubiquitous computing, but it was the UNIX line and TCP/IP that spawned ubiquitous *networked* computing, and Wintel was late to that game.

Ubiquitous Networked Computing

TCP/IP came from the UNIX timeline in 1973, but a disruptive event warping both timelines and certainly a major force setting them on a collision course was the advent of the World Wide Web, which became a major force in 1993 with the release of the NCSA Mosaic Web browser. The World Wide Web and associated browsers undoubtedly brought the Internet to the masses and made networked computing truly ubiquitous.

With PCs encroaching on the server market, Sun Microsystems made its great play, a play that was wildly successful as a technology, but not as great a play in terms of business. The play was centered on the Java programming language. Java began as an effort to create a hardware platform for interactive television, with Thompson-Sun and BroadVision (an effort with which the author was involved) but the trend toward interactive TV turned out to be a false trend as far as the mid-1990's were concerned.

What Sun Microsystems did do was some amazing *jujitsu* with Java: they flipped it from an interactive TV idea to a proposed foundation for a platform designed to counter the Wintel world. Java was essentially given away which helped its deep adoption in industry, with Sun emulating to some degree the Berkeley Software Distribution philosophy of the early UNIX days. Java was originally targeted toward interactive applications and Web browsers, with a "write once, run everywhere" motto. However, it ended up becoming most important for back end server systems and a top server technology through its J2EE platform.

Unintended Consequences and Open Source

Java and its evolution into J2EE created a standardized network computing platform which, at first, made enterprise software companies a lot of money. However, the Open Source community, whose philosophy was set in the UNIX timeline at the origin of GNU and its General Public License, was able to embrace this free and open

platform. The culture set up by GNU migrated into the Java world, and a new threat—at least from a business standpoint—emerged: the commoditization of enterprise software and software utilities. JBoss, a free J2EE server implementation, began encroaching on IBM's profitable Websphere and BEA System's profitable WebLogic.

Sun Microsystems wanted to defend against Wintel; it certainly did not intend to set off a wave of commoditization. However, this is exactly what happened: they essentially subsidized the open source community by giving away their software technology. One could argue that they made a grievous mistake of the kind Bill Gates would never make: treating the software as "free" to sell a hardware platform instead of the other way around. Of course, what brilliant strategy it would have taken to get Java adopted as a more proprietary platform is difficult to say, especially in a market where Wintel was already ascendant.

Timeline Collision and Fallout

Moore's Law, that transistor density will double essentially every 24 months, guaranteed that the cost of computing hardware platforms would decrease drastically. What no one predicted is that the software has also dropped shockingly in value, both in terms of operating system (Linux is free) and newly commoditized enterprise software. The central element of the collision: Java and the UNIX / Open Source world inspired the inexpensive software, *but the software runs on inexpensive PCs.*

Meanwhile, the hardware was created by the Wintel architecture, but the ascension of Linux may encroach on Microsoft Windows' dominance of that software. Because of the aggressive Open Source community, able to use Linux and the open networked computing platform of Java/J2EE, development tools and platforms are becoming free or very inexpensive. Once-profitable middleware, including application servers and Web servers, is losing its value as Open Source projects offer fairly robust and free alternative versions.

The Resultant Software Product Black Hole: Commoditization

Unfortunately, in the short term at least, the collision of the two timelines has created what is hard not to see as an "everybody loses" scenario—at least, everyone with a profit motive loses. Cheap or free platforms, cheap or free software tools, and even cheap or free enterprise software creates a drastic downward pressure on the price of what were once highly profitable software product segments.

It can be aggravating for a business person to see: Why are all these people giving away their labor? Why are they ruining the profit motive, do they not have to feed their families, do they not want to save for their future, buy a better car? Yet this is the reality and a difficult one for software product managers to deal with.

The truth is that some Open Source advocates truly are odd ducks with ascetic tendencies, such as Richard Stallman, of whom MIT Professor Michael Cusumano once said, "was content to live in essentially a closet." A more general truth is that the economic value of these software product profit-destroyers' efforts does not simply vanish, but is converted into reputation, community esteem, and eventually back into dollars as many of the elite developers accept high paying contracts and corporate jobs with companies smart enough to appropriate their talent. Profits in an open source world revolve more around network effects of massive and free distribution, followed by the selling of supporting services and perhaps proprietary "add-on" software.

A much deeper examination of the effects of Open Source on the software market, considered a source of new service opportunities by some and catastrophic for the software market by others, is a worthy topic but beyond the scope of this work. Understanding the key fact that Open Source is here to stay, and puts downward pressure on especially enterprise software prices, are the key concepts for the Product Manager to understand and accept. Industry titan IBM, powerful bellwether for the direction of the market due to sheer size if nothing else, has embraced Open Source and the viability of the Linux movement.

Appendix B: Industry Practitioner Interviews

Scott Case, Atlas Solutions

DATE: 02/03/2006
SUBJECT: Scott Case, Product Manager
COMPANY: Atlas Solutions, a subsidiary of the aQuantive Group

- **How do you describe the role of Product manager in your present position?**

Chameleon Technologies – wireless, Seattle "Product Marketing Strategy" Jeff—6 months, been working as a generalist on competitive positioning, market analysis, not product specific.

I own what happens to software in one of our four major product areas. It's called the Atlas Media Console. This is a tool that allows traffickers and planners at advertising agencies to set up, deliver, and track performance of online advertising. "Adjacent space" WebTrends, more of a reporting / analysis tool—how do they navigate the site. "Uneasy neighbors." Potentially complementary players.

Atlas is the technology provider, behind the media console, Atlas has 6 data centers around the world that do the image store / ad delivery. Does data processing to prepare results for campaign. They pay for ad serving on a volume basis.

The Product manager decides what features to add to the product. He or she will work with one or two Program managers to prioritize the features, and will write a Business Requirements Document to give an idea what should generally be in the release. The Program managers work with developers and Project managers to kick back schedule estimates. As needed, of course, there is a feedback loop to clarify and change course during a project.

- **How does the actual position differ from your imagined perceptions before you started?**

More or less what he expected. Bit more on the business development. Here are customer needs, and I see that development can do maybe 10 of 300 desired features. I go out and meet with the top 25 clients; I have met with 15-20 of the top 25 clients,

phone calls with others. So direct discussion. Feedback from a high level to a low level. Account Directors are responsible for managing the strategic relationship with large clients. Atlas has 2 specific documents, 1 is a strategic development priorities document. 10 large projects on that list. 9 – 12 months per project. "If we do them [set of interrelated, strong features as opposed to fragmentary, incremental improvements] we can come out with a big press release." Second document with a bunch of feature enhancement level requests, 1 – 30 day mini-projects usually single developer. Plan to the 1st list, such as redesign work flow. Smaller list, does the large project require feature enhancements on the smaller list? Hard to productize / advertise small "feature creep" enhancements. Completing a coherent set of feature upgrades composes a major project that can be advertised.

- **Do you think PM is really one role or multiple roles which have not been thoroughly differentiated in the software industry?**

Position not cohesive. Wear many hats. Once you've proved you can handle the basis, and then you can do lots of things. BASICS: Translate customer needs, 300 to 10, which ones to do and why? Pick next ones, detailed business requirements. Work with Program M's to drill down to the details. Then work with the outbound marketers, how to price, market and sells.

When told about one MSFT PM's definition of the role (Bill Shelton): Scott feels that each PM at Microsoft has a subset of activities, the title is just broad there. You're very focused at MSFT, with one small sliver of the world. MSFT is transitioning to more of the traditional corporate PM versus software product management. Virtually monopolistic control enables them to "slow the clock speed" [of the market].

- **How do you think PM in software differs from PM in traditional companies, for example, product management of Gillette Foamy or Crest Toothpaste?**

No experience with non-software project management. "Clock speed is faster", industry moves very quickly. "Web Side Story" acquired Visual Sciences, changed the road map for what's going on in the industry. Google is a huge competitor, started giving away a site analytics tool called Urchin. Productizing a feature set so that you can market changes as an actual coherent product upgrade is another challenge. In software, cannot be a perfectionist with incredibly tight 3 month project cycles. Product management is "inbound" whereas Product Marketing is "outbound". Product Marketing manager or Marketing Communications – outbound.

• What do you think is the most important PM challenge in the particular case of software products?

People want 300 features and you can only do 10. There are difficult tradeoffs in implementing new features. "Who are you going to piss off who are you going to serve?" Tradeoffs involve "keeping the lights on" versus innovation. We serve 110,000 ads per second. 6 billion ads a day. SLA (software license agreement) says you must respond within 2 milliseconds. 99.9999% uptime. Never goes down. Must dedicate enough resources to keep the lights on 24 x 7. Some of the innovations they do alter the way the infrastructure does what it is doing. So new features have a deeply cascading effect into the system which is tricky considering you have to keep the thing up 24x7. Finishing up a project next months which had 5 separate development teams, some front-end, some more focused on database engineering, some focused on impact to ad-serving infrastructure, some focus on post-service data processing procedures. Last focused on changes to reporting tools. All 1 project. Roll out one data center at a time. Try not to impact the ad serving. Development leads use SCRUM methodology, and work together. SCRUM meetings are internal, but then 1nce a day 15 minute meeting of leads. Most of the 5 teams are in Seattle. Bullpen area in XP pairs. There is 1 outsourced development team in India, but the software modules must be very well defined. Shipping software CD's. But the outsourcing hasn't worked out well. We seem to have difficulties making outsourcing work. It is very hard to carve off modular tasks.

• **Do you feel Atlas is a software product company? Or is it a hybrid of products & services? If so, what percentage do you feel is products and what percent are services?**

Atlas is probably an even hybrid play. A ton of client services / trafficking work. 50 / 50 is Atlas Solution. A client must buy a big service component when they buy product. In the U.K. it's probably 60 / 40 services / software. Within aQuantive, it's probably 80 / 20. Drive Performance Media's case, sell online advertising space. AvenueA does the whole ad agency thing. "Drive PM" is an ad network.

Philip DesAutels, Microsoft

DATE: 04/28/2006
SUBJECT: Philip DesAutels, Academic Liaison
Formerly Product Manager, Web Services
COMPANY: Microsoft

- **How do you describe the role of Product manager in your present position?**

Former position: Web Services, a product that shipped internally. Managed a series of technologies, making sure they were in sync with industry and customer's demands. Made sure the developers were developing what the customers were asking for. Web Services was an interoperability platform. Really a very technical marketing role.

- **How did you interact with the developers? Program managers?**

There is one General manager & several Program managers. There's a ship date, more-or-less set in stone. Series of multiple product life cycles. I was driving "indigo", a code word for Web Services technologies, standards called WS-*. Post-SOAP technologies, involving authentication, workflow, and security. Bill Shelton's is a superb definition of Product management. Depending on the product, might be talking to the press. I was talking to the press and being quoted in 6-7 articles a week.

A reporter talks to PR firm, PR firm sends them to Product manager, the PM brings in a developer. Set up a set of messages you want to go out. Work with developer to make sure they're shooting for the image you want to go out. For instance, Web Services are an interoperability play. As a role it's a very hard role. Real developers tend to write you off as irrelevant if you're a pure Marketing / Finance person. Having some empathy, understanding, and credentials in software is helpful.

Once you have credibility with developers, try to show something off, demo something that might be a year away, then maybe someone stays up all night to make it happen. Helps make things happen.

- **How does the actual position differ from your imagined perceptions before you started?**

You tell somebody you're hiring to be a PM. It's like the elephant and the blind man. Everyone tells you it's something different. 3 or 4 different PM's had their role very focused on their particular product. The role sucks you in, a bunch of type A people all trying to be more type A than the next one. I'm hardcore type A, about 25% of the people were just totally insane type A. I am a good connections person, connections between people, facilitating connections. Connecting up various groups was important for Web Services. Fluid job. My wife is of German descent, she's a Project manager: Product management is much more fluid and adaptable as opposed to Project management, which requires total organization of detail above all else.

- **Do you think PM is really one role or multiple roles which have not been thoroughly differentiated in the software industry?**

At some point, no matter what the process is, there's a little box where it says, "a bit of magic happens here". At the end of a car assembly line, there's a group of people called the "fixers" who do a lot of ad hock fixes. They might have hammers, 2x4's, pillows. A big part of Product management is the "fixer" job, where the magic happens.

PM could be further divided, but you might lose the magic that keeps it going. You might cut the magic in the box in making 2 smaller boxes.

- **How do you think PM in software differs from PM in traditional companies, for example, product management of Gillette Foamy or Crest Toothpaste?**

At P&G I would guess they have an operations research, or industrial engineering programs. In manufacturing, you have engineering. Development in software is art. "I've built transmissions for 20 years; it will take me 4 months to design a new one." Design is more incremental in traditional electro-mechanical manufacturing.

I was once sitting next to a polymer guy at a multi-cultural engineering event. He says, I kick six products out a year. He knows it takes 2 months from chemical to product. Biotech will much more like software in terms of being non-deterministic.

- **What do you think is the most important PM challenge in the particular case of software products?**

Alignment. Alignment of people, of stakeholders, business stakeholders, strategic stakeholders. Different parts of the companies depending on you or pushing you. Aligning those people together, getting those people to come together. Even if a group is outside of the company, you have to get them to align with your interests. It's difficult to distribute the job and has to come onto one person.

Before Microsoft, I was CTO at Erio, content based image retrieval. Used Extreme Programming to manage multiple different requirements. I would write a scenario, developers would give what features could be done this month.

• **Do you feel Microsoft is a software product company? Or is it a hybrid of products & services? If so, what percentage do you feel is products and what percent are services?**

Hybrid of products and services. 10% hosted services, 10% products that cross the boundary completely—Microsoft Word, shift F7 is Thesaurus. Allows you to go to a series of reference books not on the machine. Groove.net is a peer-to-peer collaboration tool. Use it to work on multiple documents with versioning, for instance. 80% products like OS and shrink-wrap. Microsoft consulting (MCS) is a single digit percentage. Technical support is close to 10%, but that's not really professional services in the IT sense. "Services aren't innovation. They're a way to make money."

• **I have explained the strangler tree problem of system complexity, and of having no underlying conceptual model in some software, including some Microsoft software. How does Microsoft address this?**

5 years ago I would have completely agreed (that Microsoft had problems with complexity and underlying conceptual models). I used to work at the W3C. The early Web was a beautiful thing around the time of HTML 4.0. It had sufficiency but not complexity. No cascading style sheets yet. We really had a base model. The next tier got really complicated. The second generation you had CSS, a bunch of XML, dynamic HTML, all of the scripting languages living on Web pages, a very different place from 1st generation.

We're heading toward 3rd generation now. One generation is embodied in things like Flicker and Delicious. These are sort of back to basic building blocks. Facebook is another one. "I'm not going to be the everything site." Specialized. Delicious: I'm nothing more than a bookmark site. You can tie some Flicker and some Delicious together with RSS. You can build a third product or a portal from those components. Similar to the way people used Web sites five years ago.

Another fork it's yet again even more complex. Interactive Television. Super complicated Web sites, being able to drag-and-drop stuff. Deliver a full desktop application in the browser. Even richer UI through the Web.

I am being driven toward that complexity, but the complexity comes at a big cost. Working on Web Services made me lucky. One of the fundamental problems Web Services addresses is the very problem we're talking about. Flicker and Delicious create components of a loosely coupled system. So Web Services are a way to address the complexity of the Web. It can be impossible to measure the complexity cost. Loosely coupled systems with well described interfaces are the key. Flicker & Delicious are greatly componentized systems, and are heavily used.

If you look at mail, MSN mail, gmail, etc. This is a fundamental component that has been effectively standardized. Services components in general are a very good way to reduce complexity, Web services or otherwise.

Shuman Ghosemajumder, Google

DATE: 05/11/2006
SUBJECT: Shuman Ghosemajumder (Sloan MBA)
Business Product Manager, Trust & Safety
(Formerly worked on AdSense)
COMPANY: Google

- **How do you describe the role of (Business) Product manager in your present position?**

Any Product manager is the focal point of the product line. In a matrix organization, you don't have people directly reporting to you. You are the CEO of the product without the direct-line authority. You lead a cross-functional team.

- **How does the lack of direct line authority affect your job?**

Theoretically the lack of authority is a difficulty, but practically it works pretty well. Of course it is a drawback that you don't have direct-line authority. On the other hand, you are the leader of that team, and depending on how much respect and practical authority you have, this makes a large difference. When you have a diverse cross functional team, you can leverage the strengths of the various organizations to achieve your goals. At Google, there is a general sense of cooperation and leadership; people tend to acquiesce to your requests (as the Product manager) because you have established credibility and trust.

- **How does being a non-technical, or at least a non-coder, affect your job?**

As a non-coder, you need to work very hard and show the value that you add to the team, and make the team better in tangible ways. One way you can do this is to provide the team with greater visibility and access to upper management and the rest of the corporation. Make the product successful with customers. Groups can kind of "go rogue" and get accidentally isolated, although this can also happen deliberately in experimental groups that need to have time to develop something in semi-isolation. At a company as large as IBM, you can end up with such isolation that you have fairly serious

duplication of efforts. Google tries to be apolitical and focus on the good of the project. There's always a challenge dealing with something new and entrepreneurial. Each group working on a similar product, when you merge these groups, it brings forth new ideas.

- **Does Google split up the role of Product manager somehow? How does this split between Business and Technical Product management work?**

Yes, the role is split. Google has Product Managers and Product Marketing Managers. Business Product Manager is sort of in the middle. Can lean toward one or the other (business versus technical). Business Product Management—more of a traditional business background. PM's used to be extremely technically oriented (early in Google's growth), used to be all former developers, or people with Ph.D.'s in Computer Science. Business Product managers came from more of a business background, such as management consulting. Product Marketing Managers are closer to the traditional Proctor & Gamble role. PM's / BPM's generally focus on inbound marketing, but have some outbound responsibilities as well.

- **How does the actual position differ from your imagined perceptions before you started?**

When I started, I was not sure what PM was. In my career, I have come full circle. Much earlier in my career, I was a Software Engineer and PM for a smaller product at Groupware in Canada, a real-time cooperative desktop publishing environment. Later worked with McKinsey & IBM. In my startup days, I didn't think of PM as a specific job. As a new PM at Google, I scrambled to do a bunch of research to find out what PM really was/is. What I found out is that it isn't a well understood area, and means different things in different organizations. Now I understand it most clearly as being the leader of a specific product team at Google. So leadership is the key to the role.

- **Do you think PM is really one role or multiple roles which have not been thoroughly differentiated in the software industry?**

Google has differentiated the roles to some degree. Business Product Managers, Product Marketing Managers, and simply Product Managers are all separate titles. These

career tracks are on parallel paths, although there is a tremendous amount of overlap. Product Marketing Managers are probably the most well-defined in terms being focused on outbound marketing. Associate versions of each exist, for new college grads.

- **How do you think PM in software differs from PM in traditional companies, for example, product management of Gillette Foamy or Crest Toothpaste?**

First, what's the distinction between that kind of consumer packaged goods management and high-tech? In software, it is a lot more important to focus on customer feature requests. In high tech products, there aren't a lot of features as in toothpaste. For instance, customers might like baking soda or other whitening agents, but then it's really about market segmentation. In the software industry, it is much more important to maintain feature parity with competitors. However, Google's strategy actually deemphasizes feature parity relative to other software product companies. We think in terms of "what can we do that is new and different?"

The difference between general PM vs. PM at Google has to do with creating brand new technologies. At Google, there's a lot of working with engineering and figuring out what is the brand new great technology they've managed to invent, and figuring out how to productize it. For instance, someone has invented a way to efficiently store 2 Gigabytes per user online, how do we turn that into a product? Google is a very technology driven, engineering-out kind of company.

- **What do you think is the most important PM challenge in the particular case of software products?**

Usability. There is so much complexity associated with computer applications. Software is like traditional engineering but zero marginal distribution costs. In software you can create a "million mile long bridge." (Possible, but impractical and useless.) Creating something that is actually usable becomes very tricky. IPod, for instance, has 80% market share with a small, elegant feature set as opposed to the everything-but-the-kitchen-sink approach. It's about organization, not just limiting a feature set. There will be a market for very advanced tools, but make the day to day tasks as fast as possible and allow novices to pick it up right away.

As a PM, I try to bring the perspective of the common user back to the teams. I draw comparisons to analogous products, conduct usability testing—bring the product in front of actual users. Constantly repeating that usability is the most important thing to do, and why. Making that a core value. Let's create features, technology, and then worry about usability? That doesn't work. The biggest trap technical teams fall into is creating products that are internally consistent and make sense to technical people, more and more specific, worst case, that make sense only to people who work at a specific company. Or products that make sense to technical people but not to non-technical people. This affects everything. Sometimes very technical terms become part of the general language. We make people understand words like MP3 or "RSS feed". With the term "RSS feed", you have to explain for 5 minutes, if you're talking to someone who works for the legal profession, what this means. Why can't we just say, "Subscribe to a Web site?"

What language you use is important, how you organize things. How many buttons do you have on a mouse? How do people interact with an application? How about consistency, such as menus?

- **Do you feel Google is a software product company? Or is it a hybrid of products & services? If so, what percentage do you feel is products and what percent are services?**

Google is pretty much 100% software product company. But about 90% of the product is delivered as Web-based services. We don't do any significant consulting or selling of programming services.

1. I have explained the strangler tree problem of system complexity, and of having no underlying conceptual model in some software. How does Google address this?

The way Google prevents having a negative consequence like this is by putting the responsibility for having a clean, scalable model be a responsibility of the Technical Lead. A senior engineer would work with the Product manager to maintain this clean, scalable model; this underlying conceptual model. A Tech Lead would always be more technically qualified than the other engineers on the team.

Google does not generally have Program managers (this title does exist but is not part of the regular organization the way it is at companies like Microsoft). Product managers fulfill the role Program managers fill at Microsoft. This makes the role of PM more of an overloaded role. Rather than segmenting out the specific functions, hire more Product managers for a specific product and give autonomy to specific features and subsections of the product. Keep the role as nexus of cross functionality but have an increased number of "nexi".

In terms of work life balance, some people work all the time, I am on email basically all the time, sometimes responding to emails at 2 a.m. I spend time at home working from there, spending time with the family. People don't work too much on weekends. Travel is proportional to your interest in doing it. There is a lot of flexibility to find a mode of work that works well for you. There isn't a "policeman on the corner" mentality, as long as you're achieving your goals on a quarterly basis. It is about achieving goals, not face time for the sake of face time.

2. How did the huge public IPO and all the recent publicity affect your job, or the culture at Google?

Not a lot of change after IPO. Google behaved like a public company before IPO, driven by quarterly results. Were always driven by a need to deliver value to users. Never thought of it in terms of competing with feature sets. No pressure to create strategies around competitors. Whenever Google does something, they do something to deliver great value to users and hopefully it is different from competitors. It is very important at Google not to simply one-up competitors. The opposite of the fast-follower philosophy. Still, it is important to be responsive to the needs of customers.

3. Any final comments you'd like to share about Product management?

I've learned PM is all about your individual character. One thing I heard about PM was to always remember as a PM that you are the one person on the team that is

completely disposable. If you always remember that, keep your eye on the ball of adding maximum value to team, then you will be successful.

Appendix C: Interview with Professor Daniel Jackson

DATE: 04/06/2006
SUBJECT: Daniel Jackson
Professor, Department of Electrical Engineering and Computer Science
SCHOOL: The Massachusetts Institute of Technology

Biographical Note

Daniel Jackson leads the Software Design Group in the Computer Science and Artificial Intelligence Laboratory at M.I.T. He received an MA from Oxford University (1984) in Physics, and his SM (1988) and PhD (1992) from M.I.T. in Computer Science.

Note on Interview Transcript Censorship

Professor Jackson spoke freely, informally, and allowed this researcher to use a recording device. Certain quotes involving commercial concerns and military agencies have been censored or "anonymized." Significant affected quotes are noted. Words and phrases replacing anonymized content appear [*italicized in brackets*]. Words and phrases not actually spoken but added for clarity appear in [brackets] but are not italicized.

TRANSCRIPT FOLLOWS:

[Informal introductory chatting]

PROFESSOR JACKSON: [Alloy modeling language] allows you to analyze models in isolation. You never have to actually analyze the code. It helps you to debug your thinking about the system.

JOHN HEMPE: Right, I understand.

PROFESSOR JACKSON: The second respect in which it is not verification is that all existing verification systems would make you actually prove theorems. And the sort of fundamental premise of Alloy was that what the modeler needed was a sort of push-button automation that you'd build your model and say OK, is that right?

JOHN HEMPE: I took the [Nancy] Leveson course on Software Engineering, and the essence I got from her class was that formal verification doesn't work. And there's still some argument about lightweight formal methods, but I've never seen it in industry, at least not in the enterprise software industry. Maybe in some fly-by-wire jet fighter software or something, but it just seems like the methods themselves are too complex for average professionals.

PROFESSOR JACKSON: You have to remember that the software industry, by and large, is extraordinarily conservative. Which is very surprising for a bunch of people who consider themselves radicals.

JOHN HEMPE: Define conservative in this context.

PROFESSOR JACKSON: They don't want to change the way they do things. They want to rely on, sort of, lift your finger and feel the wind. They don't want to consider the possibility that computers could help you design software. You look, for example, at the astonishing vehemence with which Extreme Programming has attacked the idea of design.

JOHN HEMPE: Yes.

PROFESSOR JACKSON: It is not the method. It's the idea that you could design a system. It's deeply unfashionable to suggest the idea that a system could be designed.

JOHN HEMPE: True. These methods like Agile, Scrum, XP, these tight iterative methods--

PROFESSOR JACKSON: What they're confusing is the principle of risk management in which you don't want to make decisions for which you don't have information, versus the idea that even when you're capable of making predictions and good decisions you should just blunder ahead anyway and pay whatever price it costs to fix it.

What they essentially do is all the design in the code, which is an extraordinarily expensive way to do things. In the face of all their criticisms of formal methods, it's amazing how much work these people are prepared to do

The other thing that is, perhaps not conservative but part of the same stubbornness, is this idea that, well, why think about requirements, because they change? Right? Now, I don't believe this. And the reason I don't believe this is the following…[although] requirements shifts are a huge source of complexity in system development, the XP people are working, by and large, for very conservative companies..

JOHN HEMPE: Ah, I did not know that.

PROFESSOR JACKSON: My guess is that the reason the requirements changed is because they never bothered to really understand. I'm being very polemical now. But I think what often happens is, from the software engineer's perspective the requirements have changed. But the [business laymen] on the other side says, "How did you ever not understand that? What did you think we were doing?" And they're amazed at the idea. [They're thinking], "We didn't change our requirements, we always wanted to do it like this." So, there's just a reluctance to spend time with the customer and truly understand what their problem is.

JOHN HEMPE: Right, right. And that's definitely part of what I'm exploring. A typical quote from an average coder I know is, "I've never met a competent marketing person." It just seems like somewhere the pipeline of actual requirements to engineers who write code gets broken down, and then the

engineers get blamed for everything, or [perhaps I should say], the programmers get blamed for everything.

PROFESSOR JACKSON: Right, exactly. That's exactly what happens. And one of the reasons is that the initial attempt to characterize the problem is not sufficiently appreciated as a technical problem. It's thought to be something that needs less technical expertise. But it needs *more* technical expertise. It's the essence of the whole enterprise: figuring out what you're trying to do. So, that, I think, is a large part of the problem.

If you look at something like alloy, or formal methods, in its purest form, the actual proposal is a very simple one: at the beginning, you should figure out what problem you're trying to solve, and you should have some fairly precise characterization of that problem. Now, you can characterize that on a very wide spectrum. At the most economical end, in a form which I think is economical for almost every software development [effort], you simply document the state of the system. You simply say, in formal terms, this is exactly the set of states I expect the system to be in.

Now, people are shocked at this idea, and if you said to them, well, here's a formal language for doing this, they'll throw up their hands and say, "I cannot possibly do this. This is absurd. I'm not going to use Alloy, or VDM, or Zed, or whatever, to specify the states of my system."

But, on the other hand, if you show them SQL and asked them if they'd write a database schema, they'd say, "Oh, I'll write a database schema, that's no problem at all."

JOHN HEMPE: Which is essentially a formal method of specifying--

PROFESSOR JACKSON: The identical thing. It is a formal description of the defined states of the system. And in fact, many people write integrity constraints over their database, which are invariants!

JOHN HEMPE: I'm trying to think of this from a purely professional, pedestrian point of view. [The writing of database schemas] is known as a highly leverageable skill. That particular tight skill of doing SQL schemas and verification, you can get hired at hundreds of companies, from Oracle on down.

PROFESSOR JACKSON: Perhaps what this means is the people who build database systems, they know they need the schema because their application generator, their transaction processor, or whatever, is going to have to take that schema as input. No schema, no system! It's as simple as that, right?

JOHN HEMPE: True.

PROFESSOR JACKSON: The people who are building applications that don't require SQL schemas perhaps don't realize that they need a schema too. And the people building the database know that, if they don't have a schema, it's not just that they can't construct a database. They also won't have a basis to think about transactions they're going to run, or how the system's going to evolve over time. They don't have a way of saying, well, our business model is truly reflective of the software we're constructing. So, for them, the schema is like the first concrete representation that there's actually some understanding of what's going on in the problem domain.

My contention is that it's really not so radical to say, whatever kind of system you're building, even if it doesn't have a database in it, you really need a schema of some sort.

JOHN HEMPE: Or you'll create one implicitly as you code.

PROFESSOR JACKSON: And the kind you'll create will be one full of strangler trees. That's the problem, right? And, so, what happens in my mind when you create such a schema is, it's extremely difficult and painful, because you're doing real work, right? And what you should be forced to do is...one of the things a formal language makes you do, whether the language is Alloy, or SQL, or even UML is, it doesn't give you any corner to hide. (Actually, UML gives you some.)

But, a real formal language gives you no corner to hide. You've got to write it down, right?

JOHN HEMPE: Right.

PROFESSOR JACKSON: What that means is, you can't punt, you can't say, "I'll figure it out later." You've got to actually commit to something. The reason that's good is, when you actually write down what you think the situation is, it is a horrible mess. Then you take a step back and say, "Now this is a real mess. It can't be like this." In the worst case, you have to go back to the problem domain and say, you know, our organization is so complicated that we've got to simplify things before we try to support it with automation, right?

But in the more common case, you say, "There must be a simpler way of viewing this." And you come up with some kind of abstraction or some way of thinking about it that generalizes appropriately or simplifies. Or you say, "You know what, I thought of the system structures being like this because I had in mind all these complicated functions, but you know what? We don't need to support those functions in our first release. We should have a simpler model, and with this dramatic simplification we can support 90% of the functions we want to support at 10% of the cost in the schema. So I think that's a tremendously valuable activity…

One other thing. There's a very positive part of Extreme Programming. It's this sort of nose-to-the-grindstone aspect. It's that, if you're not going to write schema but if you write code, you're writing Java classes, you're writing a schema in Java. What they're really saying, in a way, and I say this in the introduction to my book, it's very similar to the Alloy approach. From Day One, start dealing with the details.

My gripe with Extreme Programming is that the only way they know of to deal with the details is by writing code, but there are much simpler languages than code.

JOHN HEMPE: I see.

PROFESSOR JACKSON: Alloy, to a novice, might seem complicated, but semantically it is enormously simpler than any programming language. I mean, basically, all you've got is sets and relations, that's it. It's like a database schema but…actually, it's basically the same as a relational database schema.

JOHN HEMPE: I still feel like you'd need an "Alloy for Dummies," <chuckles>. "Here's the set theory you need, here's what SAT [the Boolean Satisfiability Problem] is…."

PROFESSOR JACKSON: You shouldn't need to know SAT. SAT is just a background thing, part of the engine.

JOHN HEMPE: I think part of it is getting something well known and [professionally] leverageable enough. Back in 2001, I knew that, if I became a Sun Certified Java Programmer, then N number of companies would hire me, and any sort of formal language or true design discipline beyond knowing just basic UML nomenclature, it wouldn't get you hired as a professional.

It's a matter of showing the power, and showing that an average professional can think on that level. And I sometimes wonder if we sometimes just run into cognitive limits where, it's really hard to think in these advanced ways.

PROFESSOR JACKSON: But it's just abstraction, right?

JOHN HEMPE: Absolutely. Abstraction is the most noble, but also the most difficult form of thinking.

PROFESSOR JACKSON: I agree, I think you're right, but why should you think the average professional would need to do that? Why isn't it just like the schema designer?

JOHN HEMPE: The Architect.

PROFESSOR JACKSON: Yes, the architect. It's going to be a smaller group of people who do that kind of work.

JOHN HEMPE: Right, right.

PROFESSOR JACKSON: And the rest of the people will do much more concrete work, dealing with the details of the code.

JOHN HEMPE: So the important thing is to have a few key experts who are able to specify the system in this formal language.

PROFESSOR JACKSON: I think that is essential.

JOHN HEMPE: I agree, in principle.

[Significant quote censored at Jackson's request]

PROFESSOR JACKSON: I think that what you find is that, in the really successful systems, in the most successful systems, at some point, someone has to come in and really understand what's going on. There's this sort of very modern belief that you can have fully decentralized, organic systems in which no individual person really understands what's going on and the whole thing can work. I don't know how far that can honestly go. I think, in the end, that the edifice crumbles. Or you just get fed up with all the rough edges between the parts.

JOHN HEMPE: So you did it, to some degree, though, in trying to rewrite the 80,000 line CM of the Air Traffic Controller engine. OK, well we'll just understand this part, and rewrite all that, and then everything will be better.

PROFESSOR JACKSON: We did that [as a research project], but it would never have been any use in context of the system as a whole to do that. The real lesson was that if we had really understood that system we could have dramatically simplified it, the whole thing. We did dramatically simplify the piece we were looking at, but the whole system could probably have been simplified.

I think one of the things that you need is a very simple notion of what problem you're trying to solve. As I get older, I become more and more narrow minded about the importance of simplicity. You always hear people say, in industry, "You academics don't understand. It's not that simple."

JOHN HEMPE: Right.

PROFESSOR JACKSON: And I'm becoming more and more unrepentant about this.

JOHN HEMPE: "You business people don't understand that it can be that simple if you do the work."

PROFESSOR JACKSON: Yes. I won't say that it can be that simple in that you can obtain the same level of functionality, maybe you can't! It might take a lot of work to do it. Yet when you look at the most successful things, often it was really simplicity.

JOHN HEMPE: Yes. My personal theory of why the Web works has to do with the fact that we're basically intelligent apes and the Web uses ostensive definition and pictures, and we're really good at processing visual information and ostensive definition—point at things.

PROFESSOR JACKSON: It's not just that, it's that having a human in the loop of every Web transaction means that you can tolerate programs that are incredibly unreliable. The system can tolerate all kinds of unpredictable things.

[The following is anonymized at Jackson's request]

I've head that [*one company's*] attitude is that basically, the Web site can do anything wrong so long as the actual transaction of buying the merchandise is OK. So you...get the wrong [*nonessential things*], how much does it matter? So long as most of the time you're getting it right. And when people say, "I want that," it works. As long as they get that right and bill the right amount on their

credit card, which is probably only a tiny bit of the system. And they've got this huge edifice of stuff all around it.

The advantage of it is, most of the system doesn't have to work that well. Now, probably hospital databases are not like that. They can't afford to have a lot of stuff that isn't working exactly right.

JOHN HEMPE: Right, and similar safety critical systems.

PROFESSOR JACKSON: Right.

JOHN HEMPE: Microsoft, their Program managers are more technical than most. They invented that role. There were no Program managers before Microsoft. So they have the formal roles. They have Product Marketing managers, Product managers, Program managers, and then Developers. Well, and Project managers. But it really is kind of a pipeline in from the market inwards, as much as they could possibly do.

PROFESSOR JACKSON: But there doesn't seem to be any role for a Conceptual Designer there.

JOHN HEMPE: When you say "Conceptual Designer", I think "Architect." Let's say, "Conceptual Architect."

PROFESSOR JACKSON: The kind of thing I'm thinking of is, you're building a Word Processor, right? And someone says, "We'd like to have Sections with all these things." Where's the guy who says, "Wait a minute, this is a paragraph based Word Processor. We don't have sections."

JOHN HEMPE: But Microsoft has some of the smartest people in the world. Surely there's someone in there who is smart enough to do that. It must happen at least at a small scale on some teams.

PROFESSOR JACKSON: It does. It certainly does. But I think the question is also one of, simply, tradeoffs in terms of where you want to pay the price. [Remainder of quote censored at Jackson's request.]

JOHN HEMPE: And, they're known as a fast-follower.

PROFESSOR JACKSON: Exactly.

[Remainder of quote censored at Jackson's request.].

JOHN HEMPE: Right.

[The following quote is anonymized]

PROFESSOR JACKSON: My conclusion from that book was that one of the things that allowed them to kill [*a competitor*] is the fact that they had the guts to hack down the strangler trees and clear the forest and start again.

JOHN HEMPE: They did do that. Although I thought it was being a little generous to call it "judo" giving it [the browser] away, I thought that was definitely a "sumo" crushing technique, the ultimate price war, give it away. In that, I thought he was a little generous to Microsoft, but you're right, they certainly came up with what turned out to be a superior architecture, at least in the short term. Netscape eventually gave their source away, starting the Mozilla project.

PROFESSOR JACKSON: I'm unfortunately going to have to go to a meeting in a few minutes. We can carry on another time, though. Where have we got to?

JOHN HEMPE: Are there any, thinking in terms of the requirements engineering, the product management, the conceptual architect ideas, are there any proceedings from the IEEE conference, or any sources you want to give me?

PROFESSOR JACKSON: Well, it sounds a little self-serving, but I've done some collaboration with my father. He's written some papers I'd recommend. Some are rather technical, but some are more overview ones. He had this idea

that, when you think about requirements, you want to think about multiple problems being solved, and to think about those problems independently. I think that's quite a powerful idea.

I would say that my biggest complaint about the way people do requirements is that there's a deeper rationality. A requirements document is often a very long list of very detailed functions. This is not a rational way to say what you require. If you're going to buy a car...you would never say, "Here are my 58 requirements." You would say, "First and foremost, I want something that looks cool and has a top that comes down. Then, I suppose I should have high fuel efficiency, and so on."

Our fundamental approach, in life, to make all decisions is that we prioritize. To me, the biggest failure of requirements engineering is that I almost never see requirements documents that are prioritized. That say, "There's an awful lot of stuff in the list we're about to give you, but here's the essence. Here's what we're really trying to achieve."

Now, people, like with simplicity, have all kinds of reasons why in practice, this won't work. In practice, there are never few enough for this to be well articulated and separated. I'm sure the Extreme Programming people would tell us that you will never [before implementation] be able to assess the different risks and put values on these things. But, so what? You've got to do that. You do that in life all the time, when you do risk management and make decisions. You assign values to things, and you assign risks, using imperfect information. Sometimes you get it wrong, but that doesn't stop you from doing it.

JOHN HEMPE: And sometimes, in business, it seems like there's this temptation to jump straight to the nuts and bolts, and just have this huge bucket of nuts and bolts dumped out in a document.

PROFESSOR JACKSON: Exactly. It's not a smart way to go. I think that it's a really, really serious problem in software requirements, that people think it's all

the technical problems. Now we know that yeah, programming costs you a lot of money, but that's not the real problem. The real problem is, you know, what was—when IBM and TRW and those companies failed in this huge FAA development—what was wrong with it was they had completely unrealistic requirements. No one had a requirements review. Which they had later with all kinds of august panels, where, some experts listened to basically what the government, the FAA, had told IBM and said, "Why did you agree with this? This is crazy! Three seconds of downtime a year?"

JOHN HEMPE: Right. I read that paper.

PROFESSOR JACKSON: And I think this applies for all kinds of systems, whether information systems, safety critical systems, business systems, startup companies, everyone seems to think, "I don't need to do that work." And I don't buy it.

Startup companies ask themselves, "Well, we need to get something out in time for the VC's." But, what's the priority? What's really going to impress them? Now, some people will do that explicitly, but they won't think of that, they'll think of working around the requirements process. "I don't have time to do a proper requirements thing here. So, I'd like all these engineering people off my back, get rid of all these formal methods people. I'm a tactical guy, I'd like to figure out how to please the VC's."

That's a perfectly rational thing, if that's what you need to do. If you've decided that the first thing you need is funding, and then it's going to be a prototype you can throw away, go for it! Then you need to think about your requirements really carefully. You need to say, "What does it mean to produce something that will impress the VC's?"

JOHN HEMPE: The problem is that the prototype usually doesn't get thrown away, and it becomes the seed of the strangler tree following very soon behind it.

PROFESSOR JACKSON: That's exactly right! Then you've got to be really honest about whether you're going to throw it away or not, that's exactly right. But I think that what's causing a lot of problems in requirements engineering is a failure to apply these very basic common sense things. It's not clear to me why people aren't doing it, to be honest.

JOHN HEMPE: Right, right. I think I have some idea. I think there's a strong temptation to just get down and code, because that's what people know. That's the leverageable, widespread skill people have. Frankly, it requires a lower level of thinking. I remember reading about how Samuel Coleridge wrote about generalizing being the highest form of thought. Well, it is, but it's also more difficult.

PROFESSOR JACKSON: There's another phenomenon, I think, it's the, "Nobody got fired for buying IBM."

It's that, if you're going to change this stuff, you're going to have to be the manager who says, "Now, wait a minute. We're not going to produce any code or documentation for a week. We're going to sit in front of the whiteboard and we're going to brainstorm."

JOHN HEMPE: Which scares the hell out of those used to the usual mundane methods--

PROFESSOR JACKSON: "We're going to lose a week's productivity by drinking coffee and chatting?" You've got to be a gutsy manager to do that, right? And I can understand that. From a research perspective, we do that all the time, because we don't have anyone breathing down our necks. We have much more flexibility in terms of when we deliver value, and so on. I think that, somehow, one of the advantages, paradoxically, of formal methods and that kind of stuff, is that if you do modeling, you can demonstrate [the models] and you can produce stuff in the early phase, that is indicative of real value.

JOHN HEMPE: Something people can get their hands [minds] around and look at visually.

PROFESSOR JACKSON: This phenomenon--I should really go, and I'm enjoying this conversation.

[An extended exchange involving DARPA defense contracting was censored at Jackson's request for security and industrial relations reasons.]

JOHN HEMPE: I don't want to suck up the rest of your afternoon. You talked about your father's papers....

PROFESSOR JACKSON: I'll need to send you the ones I mentioned because it will be hard for you to find one.

JOHN HEMPE: Sure, ok. Maybe you could email me...some of your most relevant papers.

PROFESSOR JACKSON: I think, actually, although I'm somewhat skeptical about the whole Extreme Programming world. I think some of the people who write on that stuff are pretty smart. I think some of the lessons of *The Mythical Man-Month* are still relevant.

JOHN HEMPE: Yes, I've read that.

PROFESSOR JACKSON: Of course, you've obviously read that...I should send you some stuff by Martin Thomas. There's a guy who started a formal methods company called Praxis in Britain which has been very successful.

JOHN HEMPE: I've read a paper on that.

PROFESSOR JACKSON: He's a very interesting guy. Let me see...I'll look those two things up for you.

JOHN HEMPE: O.K...Well, thanks a lot for taking some time with me.

PROFESSOR JACKSON: Oh, fun to talk to you.

JOHN HEMPE: Thanks! We should talk again.

PROFESSOR JACKSON: Yes, let's talk again.

JOHN HEMPE: Absolutely.

*****TAPE TERMINATES*****