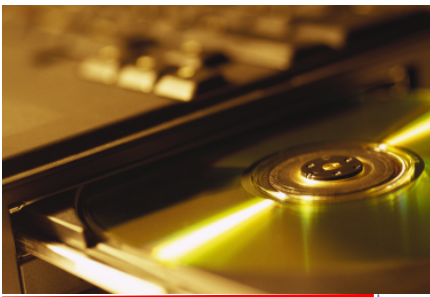




Concurrency

www.scpdnet.org

SCPD: Integrating Strategy, People, Process, Tools, and Technology



Tactics and Benefits of Concurrent Software Development ^{1, 2}

Mary Poppendieck

Programming is a lot like die cutting: The stakes are high and mistakes are costly. Concurrent development enables discovery of big, costly problems before it's too late...

When sheet metal is formed into a car body, a massive stamping machine presses the metal into shape. A huge metal tool called a die makes contact with the sheet metal, pressing it into the shape of a fender or a door or a hood. Designing and cutting these dies to the proper shape accounts for half of the capital investment of a new car development program, and drives the critical path. If a mistake ruins a die, the entire program suffers a huge setback. So if there is one

(Continued on page 2)

Boston Chapter Programs 2005-2006	p 12
SCPD Journal	p 13
Vision, Mission, Values, Objectives	pp 14-15
Boards & Sponsors	p 16
Join Our Publications Team	p 16
Become a Member	p 17
Why SCPD?	p 17

CPD in the News p 18-19

- 3D Design Software
- Innovation Labs
- Collocated teams

SCPD MISSION

To further the development of and to promote the application of Concurrent Product Development (CPD) in companies and organizations worldwide.

1. This article originally appeared under the title, "Morphing the Mold" in the August 2003 issue of "Software Development." Copyright © 2003 CMP Media LLC. Reprinted with permission.
2. Article based on Chapter 3 of the book, "Lean Software Development: Agile Toolkit," by Mary and Tom Poppendieck. © 2003 Mary Poppendieck. Reproduced by permission of Pearson Education, Inc. All rights reserved. Visit Addison Wesley at: <http://www.awprofessional.com/search/index.asp?searchstring=0321150783&searchgroup=Entire+Site&searchtype=ISBN>

Concurrent Software Development^{1, 2}

(Continued from page 1)

thing that automakers want to do right, it's the die design and cutting.

The problem is, as car development progresses, engineers keep making changes that find their way to the die design. No matter how hard the engineers try to freeze the design, they are not able to do so. In Detroit in the 1980s, the cost of design changes was 30-50 percent of the total die cost, while in Japan it was 10-20 percent of the total die cost. These numbers suggest that the Japanese companies must have been much better at preventing change after the die specs were released to the tool and die shop, but such was not the case.

The US strategy for die-making was to wait until the design specs were frozen, and then send the final design to the tool and die maker, which triggered the process of ordering the block of steel and cutting it. Any changes went through an arduous approval process—it took about two years from ordering the steel to the time that the die would be used in production. In Japan, however, the tool and die makers ordered the steel blocks and started rough cutting at the same time the car design began. This is called *concurrent development*. How can it possibly work?

Japanese die engineers are expected to know a lot about what a die for a front door panel, for example, involves, and are in constant communication with the body engineer. They anticipate the final solution and they are also skilled in techniques to make minor changes late in development, such as leaving more material where changes are probable. Most of the time die engineers are able to accommodate the engineering design as it evolves. In the rare case of a mistake, a new die can be cut much faster because the whole process is streamlined.

Japanese automakers do not freeze design points until late in the

(Continued on page 3)

Mary Poppendieck's 25 years' experience in IT include supply chain management, manufacturing systems and digital media. As information systems manager in a video tape manufacturing plant, she first encountered the Toyota Production System, which later became known as Lean Production. She implemented one of the first Just-in-Time systems at 3M, resulting in dramatic improvements in the plant's performance.

Concurrent Product Development

...is a systematic team-driven approach to simultaneously accomplish the product and process engineering design activities, and the product and project management activities, that are directly and indirectly required to define, rapidly develop, produce, test, service, and document new products. (SCPD)

1. This article originally appeared under the title, "Morphing the Mold" in the August 2003 issue of "Software Development." Copyright © 2003 CMP Media LLC. Reprinted with permission.
2. Article based on Chapter 3 of the book, "Lean Software Development: Agile Toolkit," by Mary and Tom Poppendieck. © 2003 Mary Poppendieck. Reproduced by permission of Pearson Education, Inc. All rights reserved. Visit Addison Wesley at: <http://www.awprofessional.com/search/index.asp?searchstring=0321150783&searchgroup=Entire+Site&searchtype=ISBN>

Concurrent Software Development^{1, 2}

(Continued from page 2)

development process, allowing most changes to occur while the window for change is still open. In contrast to the early design freeze practices in the US in the 1980's, Japanese die makers spent perhaps a third as much money on changes, and produced better die designs. Japanese dies generally required fewer stamping cycles per part, creating significant production savings.

This significant difference in time-to-market and the increasing market success of Japanese automakers prompted U.S. automotive companies to adopt concurrent development practices in the 1990s, and today, the product development performance gap has narrowed significantly.

Concurrent Software Development

Programming is a lot like die cutting. The stakes are often high, and mistakes can be costly, so sequential development—establishing requirements before design and programming begin—is a common way to protect against serious errors. Sequential development forces designers to take a depth-first rather than breadth-first approach to design, making low-level dependent decisions before experiencing the consequences of the high-level decisions. The most costly mistakes are made by forgetting to consider something important at the beginning, and the easiest way to make big mistakes is to drill down to detail too fast. Once you start down the detailed path, you can't back up, and aren't likely to realize that you should. Instead, it's wiser to first survey the landscape and delay the detailed decisions.

Concurrent development of software encourages this breadth-first approach, enabling discovery of those big, costly problems before it's too late.

(Continued on page 4)



Concurrent Product Development

...encompasses but is not limited to concepts such as Concurrent Engineering, Integrated Product Development, Integrated Product and Process Development, Simultaneous Engineering, Concurrent Design, Concurrent Product and Process Development, Collaborative Engineering, Cross Functional Teams, and other similar or related concepts.
(SCPD)

1. This article originally appeared under the title, "Morphing the Mold" in the August 2003 issue of "Software Development." Copyright © 2003 CMP Media LLC. Reprinted with permission.
2. Article based on Chapter 3 of the book, "Lean Software Development: Agile Toolkit," by Mary and Tom Poppendieck. © 2003 Mary Poppendieck. Reproduced by permission of Pearson Education, Inc. All rights reserved. Visit Addison Wesley at: <http://www.awprofessional.com/search/index.asp?searchstring=0321150783&searchgroup=Entire+Site&searchtype=ISBN>

Concurrent Software Development^{1, 2}

(Continued from page 3)

Usually taking an iterative form, concurrent development is the preferred approach when stakes are high and understanding of the problem is evolving. Moving from sequential to concurrent development means programming the highest-value features as soon as a high-level design is determined, even while detailed requirements are still being investigated. This may sound counterintuitive, but consider it an exploratory approach that permits you to learn by trying a variety of options before you lock in on a direction that constrains implementation of less important features.

In addition to providing insurance against costly mistakes, concurrent development is the best way to deal with changing requirements, because both small and large decisions are deferred while you consider all the options. Because of this flexibility, when change is inevitable, concurrent development reduces delivery time and overall cost, while improving the performance of the final product.

This may sound like magic—or hacking—and, indeed, merely programming earlier, without associated expertise and collaboration, is unlikely to lead to improved results. Vital critical skills must be in place for concurrent development to be truly effective.

In traditional, sequential development, U.S. automakers considered die engineers to be quite remote from automotive engineers. Similarly, programmers in a sequential development process often have little contact with the customers and users who have requirements and the analysts who collect requirements. Concurrent development in die cutting required U.S. automakers to

(Continued on page 5)

“...concurrent development is the best way to deal with changing requirements”

Concurrent Engineering

...is a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. This approach is intended to cause the developers, from the outset, to consider all aspects of the product life cycle from concept through disposal, including quality, cost, schedule, and user requirements.

Institute for Defense Analyses (IDA) Report R-338, The Role of Concurrent Engineering in Weapons System Acquisition, 1988.

1. This article originally appeared under the title, “Morphing the Mold” in the August 2003 issue of “Software Development.” Copyright © 2003 CMP Media LLC. Reprinted with permission.
2. Article based on Chapter 3 of the book, “Lean Software Development: Agile Toolkit,” by Mary and Tom Poppendieck. © 2003 Mary Poppendieck. Reproduced by permission of Pearson Education, Inc. All rights reserved. Visit Addison Wesley at: <http://www.awprofessional.com/search/index.asp?searchstring=0321150783&searchgroup=Entire+Site&searchtype=ISBN>

Concurrent Software Development ^{1, 2}

(Continued from page 4)

make critical changes: the die engineer had to anticipate what the emerging design would need in the cut steel, and thus had to collaborate closely with the body engineer.

Similarly, concurrent software development requires developers with enough expertise in the domain to anticipate where the emerging design is likely to lead, and close collaboration with the customers and analysts to design the system.

The Last Responsible Moment

Concurrent software development means starting developing when only partial requirements are known and developing in short iterations that provide the feedback that causes the system to emerge. Concurrent development makes it possible to delay commitment until the *last responsible moment*, a term coined by the Lean Construction Institute (www.leanconstruction.org); that moment at which failure to make a decision eliminates an important alternative. If commitments are delayed beyond the last responsible moment, decisions are made by default—generally not an effective method. .

Making decisions at the last responsible moment isn't the same as procrastination; in fact, delaying decisions is hard work, requiring special tactics:

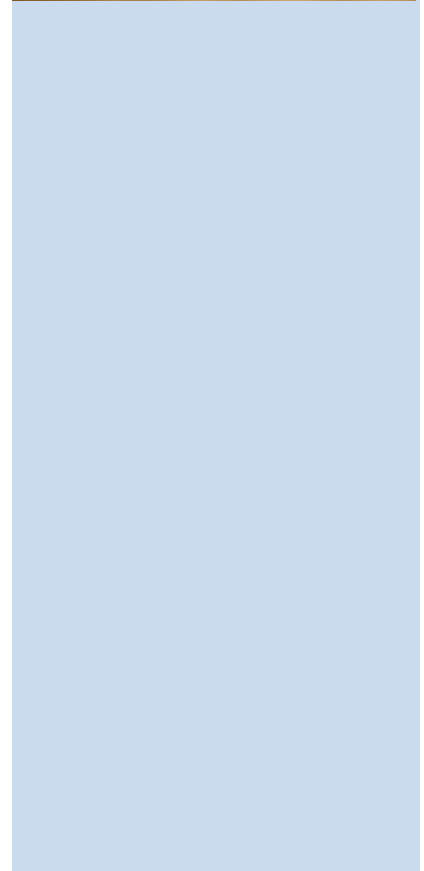
Share partially complete design information.

The notion that a design must be complete before it's released is the biggest enemy of concurrent development, increasing the length of the feedback loop in the design process, causing irreversible decisions to be made far sooner than necessary. Good design is a discovery process, best accomplished with short, repeated exploratory cycles.

Organize for direct, worker-to-worker collaboration.

Early release of incomplete information means refining the design as development proceeds, and requires collaboration: The upstream workers who understand system details must communicate directly with downstream workers who understand code details.

(Continued on page 6)



1. This article originally appeared under the title, "Morphing the Mold" in the August 2003 issue of "Software Development." Copyright © 2003 CMP Media LLC. Reprinted with permission.
2. Article based on Chapter 3 of the book, "Lean Software Development: Agile Toolkit," by Mary and Tom Poppendieck. © 2003 Mary Poppendieck. Reproduced by permission of Pearson Education, Inc. All rights reserved. Visit Addison Wesley at: <http://www.awprofessional.com/search/index.asp?searchstring=0321150783&searchgroup=Entire+Site&searchtype=ISBN>

Concurrent Software Development^{1, 2}

(Continued from page 5)

Develop a sense of what's critically important in the domain.

Forgetting some critical feature of the system until too late is the fear that drives sequential development. If security, response time or fail-safe operation are critically important in the domain, these issues should be considered from the start; if they're ignored until too late, it will indeed be costly. However, the assumption that sequential development is the best way to discover these critical features is flawed. In practice, early commitments are more likely to overlook such critical elements than late commitments, because early commitments rapidly narrow the field of view.

Develop a sense of when decisions must be made.

If you make decisions by default, you haven't truly delayed them. Certain architectural concepts such as usability design, layering and component packaging are best made early, to facilitate emergence in the rest of the design. Late commitment must not degenerate into no commitment. You need to develop a keen sense of timing that kicks in your decision-making mechanism at the appropriate moment.

Develop a quick response capability.

The slower you respond, the earlier you must make decisions. Dell, for instance, can assemble computers in less than a week, so they can decide what to make less than a week before shipping. Most other computer manufacturers take a lot longer to assemble computers, so they must decide what to make much sooner. If you can change your software quickly, you can wait to make a change until customers know what they want.

Cost Escalation

Software systems differ from most products in that they're upgraded on a regular basis. More than half of the development work that occurs on a software system takes place after it's first sold or placed into production. In addition to internal changes, software systems are subject to a changing environ-

(Continued on page 7)



1. This article originally appeared under the title, "Morphing the Mold" in the August 2003 issue of "Software Development." Copyright © 2003 CMP Media LLC. Reprinted with permission.
2. Article based on Chapter 3 of the book, "Lean Software Development: Agile Toolkit," by Mary and Tom Poppendieck. © 2003 Mary Poppendieck. Reproduced by permission of Pearson Education, Inc. All rights reserved. Visit Addison Wesley at: <http://www.awprofessional.com/search/index.asp?searchstring=0321150783&searchgroup=Entire+Site&searchtype=ISBN>

Concurrent Software Development^{1, 2}

Continued from page 6)

ment—say, a new operating system, a change in the underlying database, a change in the client used by the GUI or a new application using the same database and so on. Most software is expected to change regularly over its lifetime, and in fact, once upgrades are stopped, software is often nearing the end of its useful life. This presents us with a new category of waste; that is, waste caused by software that is difficult to change.

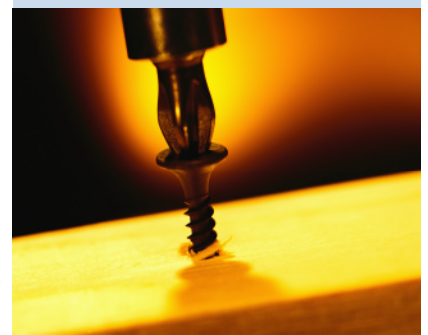
In 1987, Barry Boehm wrote, “Finding and fixing a software problem after delivery costs 100 times more than finding and fixing the problem in early design phases.” This observation became the rationale behind thorough up-front requirements analysis and design, even though Boehm himself encouraged incremental development over “single-shot, full product development.” In 2001, Boehm noted that for small systems, the escalation factor is more like 5:1 than 100:1; and even on large systems, good architectural practices can significantly reduce the cost of change by confining features that are likely to change to small, well-encapsulated areas.

Product development previously reflected a similar, but more dramatic, cost escalation factor. It was once estimated that a change after production began could cost 1,000 times more than if the change had been made in the original design. This belief that the cost of change escalates as development proceeds contributed greatly to standardizing the American sequential development process. No one seemed to recognize that the sequential process could actually be *causing* the high escalation ratio. However, in the 1990s, as concurrent development replaced sequential development in the U.S., the cost escalation discussion was forever altered. The discussion was no longer about how much a change might cost later in development; instead, it centered on how to reduce the need for change through concurrent engineering.

Not all change is equal: You need to get a few basic architectural decisions—such as choice of language, architectural layering decisions, or the selection of interacting with an existing database—right at the onset of development, because they fix constraints throughout a system’s lifespan. These kinds of decisions might have the 100:1 cost escalation ratio. Because these decisions are so crucial, you should take a breadth-first approach and try to minimize the number of these high-stakes constraints. The bulk of change in a sys-

(Continued on page 8)

**“Concurrent
design defers
decisions as late
as possible.”**



1. This article originally appeared under the title, “Morphing the Mold” in the August 2003 issue of “Software Development.” Copyright © 2003 CMP Media LLC. Reprinted with permission.
2. Article based on Chapter 3 of the book, “Lean Software Development: Agile Toolkit,” by Mary and Tom Poppendieck. © 2003 Mary Poppendieck. Reproduced by permission of Pearson Education, Inc. All rights reserved. Visit Addison Wesley at: <http://www.awprofessional.com/search/index.asp?searchstring=0321150783&searchgroup=Entire+Site&searchtype=ISBN>

Concurrent Software Development ^{1, 2}

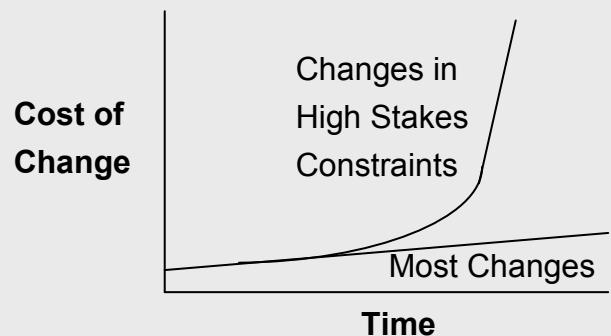
Continued from page 7)

tem need not have a high cost-escalation factor; as you move through development, the sequential approach escalates the cost of most changes exponentially. Sequential development emphasizes getting all the decisions made as early as possible, so the cost of all changes is the same—very high. Concurrent design defers decisions as late as possible. This has four effects:

- Reduces the number of high-stake constraints.
- Gives a breadth-first approach to high-stakes decisions, making correct decisions more likely to occur.
- Defers the bulk of the decisions, significantly reducing the need for change.
- Dramatically decreases the cost escalation factor for most changes.

A single cost-escalation factor or curve is misleading. Instead of a chart depicting a single trend for all changes, a more appropriate graph shows at least two cost escalation curves (see graphic). The lean development objective is to move as many changes as possible from the top curve to the bottom curve.

Let's return for a moment to the Toyota die-cutting example. The die engineer sees the conceptual design of the car and knows roughly the size of door panel necessary. With that information, a large enough steel block can be ordered. If the concept of the car changes from a small, sporty car to a mid-size family car, the block of steel may be too small—a costly mistake. But the die engineer knows that once the overall concept is approved, it won't change, so the steel can be safely ordered long before the details of the door emerge. Concurrent design is a robust design process be-



Two Cost Curves

A single cost-escalation factor is misleading. Instead of a chart depicting a single trend for all changes, a more appropriate graph shows at least two cost-escalation curves.

The lean [concurrent] development objective is to move as many changes as possible from the top curve to the bottom curve.

(Continued on page 9)

1. This article originally appeared under the title, "Morphing the Mold" in the August 2003 issue of "Software Development." Copyright © 2003 CMP Media LLC. Reprinted with permission.

2. Article based on Chapter 3 of the book, "Lean Software Development: Agile Toolkit," by Mary and Tom Poppendieck. © 2003 Mary Poppendieck. Reproduced by permission of Pearson Education, Inc. All rights reserved. Visit Addison Wesley at: <http://www.awprofessional.com/search/index.asp?searchstring=0321150783&searchgroup=Entire+Site&searchtype=ISBN>

Concurrent Software Development^{1, 2}

(Continued from page 8)

cause the die adapts to whatever design emerges.

Lean software development (*Here, lean is used interchangeably with concurrent and agile*) delays freezing all design decisions as long as possible, because it's easier to change a decision that hasn't been made. Lean software development emphasizes developing a robust, change-tolerant design, one that accepts the inevitability of change and structures the system so that it can be readily adapted to the most likely kinds of changes.

Software changes throughout its lifecycle mainly because the business process in which it's used evolves over time. Some domains evolve faster than others, and some domains may be essentially stable. It's not possible to build in flexibility to accommodate arbitrary changes cheaply. The idea is to build tolerance for change into the system in the domain dimensions that are likely to change. Observing where the changes occur during iterative development gives a good indication of the places the system will probably need flexibility in the future. The secret is to know enough about the domain to maintain flexibility, yet avoid excess complexity.

Keeping Flexible

If you build a system with a focus on getting everything right at the beginning, it's likely to be brittle and not accept changes readily. Worse, the chance of making a major mistake in the key structural decisions is increased with this depth-first approach.

If, on the other hand, you allow the system's design to emerge through iterations, it will be robust, adapting more readily to changes that occur during development. More importantly, the ability to adapt will be built into the system, so that as more changes occur after its release, they can be readily incorporated. [Employing the flexibility and delayed decision-making of the Japanese auto production methodology to your development project](#), you can increase its chances of success.

“It's easier to change a decision that hasn't been made”

Amateurs Strive, Experts Hide ...

(Continued on page 10)

1. This article originally appeared under the title, “Morphing the Mold” in the August 2003 issue of “Software Development.” Copyright © 2003 CMP Media LLC. Reprinted with permission.
2. Article based on Chapter 3 of the book, “Lean Software Development: Agile Toolkit,” by Mary and Tom Poppendieck. © 2003 Mary Poppendieck. Reproduced by permission of Pearson Education, Inc. All rights reserved. Visit Addison Wesley at: <http://www.awprofessional.com/search/index.asp?searchstring=0321150783&searchgroup=Entire+Site&searchtype=ISBN>

Concurrent Software Development^{1, 2}

(Continued from page 9)

Professor Thimbleby's rules for delaying commitment

In his essay, "Delaying Commitment" (IEEE Software, 1988), leading British computer scientist and professor Harold Thimbleby observes that the difference between amateurs and experts is that experts know how to delay commitments and conceal their errors for as long as possible, repairing flaws before they cause problems. Amateurs try to get everything right the first time, so overloading their problem-solving capacity that they end up committing early to wrong decisions. Thimbleby recommends some tactics for delaying commitment in software development, in a virtual endorsement of object-oriented design and component-based development:

Use Modules

Information hiding, or more generally behavior hiding, is the foundation of object-oriented approaches. Delay commitment to the module's internal design until the client interface requirements stabilize.

Use Interfaces

Separate interfaces from implementations. Clients should not depend on implementation decisions.

Use Parameters

Make magic numbers—constants that have meaning—into parameters. Make magic capabilities like databases and third-party middleware into parameters. By passing capabilities into modules wrapped in simple interfaces, your dependence on specific implementations is eliminated and testing becomes much easier.

Use Abstractions

Abstraction and commitment are inverse processes. Defer commitment to specific representations as long as the abstract serves immediate design needs.

Avoid Sequential Programming

Use declarative programming rather than procedural programming, trading off performance for flexibility. Define algorithms in a way that does not depend on a particular order of execution.

(Continued on page 11)

1. This article originally appeared under the title, "Morphing the Mold" in the August 2003 issue of "Software Development." Copyright © 2003 CMP Media LLC. Reprinted with permission.
2. Article based on Chapter 3 of the book, "Lean Software Development: Agile Toolkit," by Mary and Tom Poppendieck. © 2003 Mary Poppendieck. Reproduced by permission of Pearson Education, Inc. All rights reserved. Visit Addison Wesley at: <http://www.awprofessional.com/search/index.asp?searchstring=0321150783&searchgroup=Entire+Site&searchtype=ISBN>

Concurrent Software Development^{1, 2}

(Continued from page 10)

Beware of Custom Tool Building.

Investment in frameworks and other tooling frequently requires committing too early to implementation details that end up adding needless complexity and seldom pay back. Frameworks should be extracted from a collection of successful implementations, not built on speculation.

Additional tactics for delaying commitment include the following (Added by the author, Mary Poppendieck, who calls them “the ‘modern day’ version of what Thimbleby would have said”):

Avoid Repetition

This is variously known as the Don’t Repeat Yourself (DRY) or Once And Only Once (OAOO) principle. If every capability is expressed in only one place in the code, there will be only one place to change when that capability must evolve, and there will be no inconsistencies.

Separate Concerns

Each module should have a single, well-defined responsibility—so that a class has only one reason to change.

Encapsulate Variation

What’s likely to change should be inside—the interfaces should be stable. Changes should not cascade to other modules. This strategy, of course, requires a deep understanding of the domain and knowledge of both its stable and variable aspects. With application of appropriate patterns, you can extend the encapsulated behavior without modifying the code itself.

Defer Implementation of Future Capabilities

Rather than including capabilities you “know” you’ll need in future, implement only the simplest code that will satisfy immediate needs. Don’t speculate: You’ll know better what you really need in the future, and at that time, simple code will be easier to extend if necessary.

Avoid Extra Features

In addition to avoiding those features you “know” you’ll need, make sure you don’t add extra features “just in case.” Extra features add an extra burden of code to be tested, maintained and understood. Extras add complexity, not flexibility.



Society of Concurrent Product Development

Boston Chapter

Furthering the art and science of rapid product development



Industry and other professional societies recognize the Boston Chapter of SCPD as the best value and most accessible source for the advancement of product development practices in the eastern United States.



Jerry Robertson
Boston Chapter
President

"I remember my first meeting as a member of the Board of Directors of the then Society of Concurrent Engineering. Brad Goldense was explaining the enormous benefits from using Best Practices in Product Development. I thought to myself and then said, 'I'd settle for good practices.' Over the years, attending meetings and conferences, I learned the types of extraordinary effort that people have invested into their processes. This is not just a nice thing to do – companies have been rescued from oblivion because someone cared enough to make the investment."

SCPD Boston Chapter Programs 2005-2006

- **September 22, 2005:** **Are You Innovation Ready?** Assessing People and Culture
- **October 27, 2005:** **CPD for the Environment.** Coping with the ROHAS Initiative
- **December 1, 2005:** **From Skunkworks to Bottom Line**
- **February 9, 2006:** **Innovations and Nanotechnology**
- **March 30, 2006:** **Portfolio Planning and the Innovation Dilemma**
- **May 11, 2006:** **Is Innovation Market or Technology Driven?**

For the latest topic, speaker details, and directions, and reports of past meetings visit the Boston Chapter web page: <http://www.scpdnet.org/boston>

Call for Papers:
Journal of Concurrent Product Development

The launch date for our new refereed journal is rapidly approaching. Papers are solicited in all core areas of Concurrent Product Development, including:

- Product strategy
- Portfolio management
- Pipeline management
- Resource management
- Product design/development
- Product testing
- Commercialization
- Cross functional teams
- Management

***Inaugural Edition
coming soon
!!!***

To submit a paper:

Papers should be approximately 6000 words and submitted both as hard copy and by email.

Mail hard copy to Society of Concurrent Product Development, PO Box 68, Dedham, MA 02027-0068.

Please also email the following information to David Meeker at meecker@mit.edu :

- Title of paper
- Names of all authors
- Name, address, phone number, fax number, and email address of corresponding author
- Three to five keywords
- Microsoft Word version of paper to facilitate the review process

For more information, please contact David Meeker at meecker@mit.edu

SCPD BOD Updates and expands SCPD Mission Statement

At its August 3, 2005 meeting, your SCPD Worldwide Board of Directors voted unanimously to update and expand the SCPD Mission Statement. These changes consist of the following:

- Replace the terms Concurrent Engineering (CE) and Integrated Product Development (IPD) with Concurrent Product Development (CPD).
- Add two statements defining and clarifying the scope of CPD.

The purpose of these changes is:

- To make the mission statement consistent with the name change of the society in 2001 from the Society of Concurrent Engineering (SOCE) to the Society of Concurrent Product Development (SCPD).
- To more clearly convey that SCPD is a multidisciplinary organization concerned with critical issues and best practices of interest to product development professionals of all disciplines.
- To emphasize that SCPD continues to encompass the concurrent engineering concept upon which SOCE was founded, as defined in the 1988 IDA report (see page 4), as well as other similar or identical concepts such as Integrated Product Development, Simultaneous Engineering, and others.

The SCPD Vision, Mission, Values, and Objectives, revised as described above, are presented on page 15.

SCPD: Integrating Strategy, People, Process, Tools, and Technology



SCPD VISION

To be recognized by industry, academia, and by other professional societies as the best value source to attain the knowledge necessary to achieve advanced product development capabilities and practices

SCPD MISSION *(As revised 8/3/2005 by SCPD Board of Directors)*

To further the development of and to promote the application of Concurrent Product Development (CPD) in companies and organizations worldwide.

- **CPD is a systematic team-driven approach** to simultaneously accomplish the product and process engineering design activities, and the product and project management activities, that are directly and indirectly required to define, rapidly develop, produce, test, service, and document new products.
- **CPD encompasses but is not limited to concepts such as Concurrent Engineering**, Integrated Product Development, Integrated Product and Process Development, Simultaneous Engineering, Concurrent Design, Concurrent Product and Process Development, Collaborative Engineering, Cross Functional Teams, and other similar or related concepts.

SCPD VALUES

- **Leadership:** To embrace rapid product realization techniques and to advance our nation’s economy, driven by ourselves, our companies and our Sponsors.
- **Member Recognition:** To individuals in our organizations as facilitators of improvement, to our companies and to Sponsors for foresight in fostering environments that lead to the adoption of improved design practices.
- **Learning:** To satisfy our thirst for continuing personal development and renewal and to provide an accessible resource for industry as a whole, bringing new knowledge and skills to the workplace.
- **Networking:** To stay abreast of industry trends, to interact with like-minded professionals and to identify opportunities for business relationships.
- **Friendship:** To make professional acquaintances and to solidify old relationships; taking the SCPD meeting as a professionally rewarding yet enjoyable “time out” from the pace of daily work.

SCPD OBJECTIVES

- Disseminate knowledge to promote **understanding** of CPD concepts and processes.
- Provide a continuous **forum** for networking and sharing of ideas among professionals in all disciplines involved in product development.
- Improve **enterprise effectiveness** by expanding the CPD Body of Knowledge by emphasizing the implementation of practical approaches in industry.
- Participate in the origination and/or refinement of CPD knowledge using both internal capabilities and **collaborative relationships**.
- Foster a continuous learning organization by maintaining a CPD **Body of Knowledge** that remains comprehensive while focusing resources and activities on emerging and leading edge techniques.
- Operate to achieve **multi-national and multi-lingual** communications and text capabilities.

SCPD: Integrating Strategy, People, Process, Tools, and Technology

SCPD Board of Directors

President: Bradford L. Goldense, Goldense Group, Inc.
VP Member Communications: Peter J. Fritz, Six Sigma Manager, 3M
VP Membership: J. Richard Power, Power Plans
VP Internet & Webmaster: Richard W. Mason, Bayer Diagnostics
VP Member Publications: John P. Cushman, Technology Interface
VP Finance and Treasurer: Alexander J. Cooper, Management Roundtable
VP Refereed Publications: David G. Meeker
VP Knowledge Management - BOK: Curtis D. Hargadine, Development Professionals, Inc.
VP Knowledge Management - Website: Donald M. Stewart, Concurrency Group
VP Distance Education: R. Bruce Pittman, Pittman Associates
General Director: Bob Maigret
General Director: Frank Hull

Boston Chapter President: Jerry Robertson, MKS Instruments, Inc. [Retired]

SCPD Advisory Board

Stephen D. Eppinger, Co-Director, MIT Center for Innovation in Product Development
Winston A. Knight, Chair, URI Department of Industrial Engineering
Donald G. Reinertsen, President, Reinertsen & Associates
Jan W. F. Wasley, Chief Scientific Officer, Institute for Pharmaceutical Discovery
Donald Sebastian, Executive Director, Center for Manufacturing Systems, Stevens Institute of Technology
Preston G. Smith, Founder, New Product Dynamics
Gerald G. Colella, VP, Global Business Operations, MKS Instruments
David R. Holm, Director, Advanced Development, Consumer & Commercial Equipment Division, John Deere
Ralph Schmitt, President, Kysor/Warren
Hugh Vallely, Director, Product Planning, Harley-Davidson [Retired]



Join Our Publications Team

Product Development Professionals of all disciplines are invited to work with our editors and other Board Members, helping to carry out SCPD's mission to promote the application of Concurrent Product Development.

**Interesting volunteer opportunities include (but are not limited to):
Identifying content ... arranging reprint permissions ... authoring, designing, editing, and reviewing content of all kinds for our periodicals ... participating in virtual meetings to help determine and implement editorial policy, etc.**

Choose your area of interest and time commitment.

For information contact Peter at pjfritz2000@mmm.com; or John at jjcush@ix.netcom.com

SCPD Membership Information

12 months for \$50 (\$86 with EMJ)*

24 months for \$90 (\$162 with EMJ)*

12 months student for \$20 (\$56 with EMJ)*

* Subscribe to the Engineering Management Journal

Through a cooperative arrangement with the American Society for Engineering Management (ASEM), SCPD members may subscribe to ASEM's quarterly Engineering Management Journal (EMJ) for an additional \$36 per year.

Information you can use!

EMJ provides articles and features related to the management of engineering and technical professionals and their organizations. Practical and pertinent articles and reviews help readers gain insights and meet the challenges of coordinating the design, integration, and use of new technology in the workplace.

To join SCPD, go to www.scpdnet.org



Did you know?

SCPD was founded in Los Angeles in 1992 as the Society of Concurrent Engineering, SOCE

Why Is SCPD Named SCPD?

Brad Goldense (From www.Scpdnet.org)

SCPD is an educational organization centered on the values, strategies, principles, processes, practices, tools, technologies and people capabilities related to the "Concurrent Engineering and Concurrent Product Development Body Of Knowledge." The common approach to product development, still used by most industrial and high-tech companies around the globe, is based on the Taylor assembly line principle. Sequential, evolving to rapid sequential, practices sustained this approach for nearly eight decades. Corporations that achieved "rapid sequential" found that they were now approaching concurrent, at least in certain design and/or development functions. Additional research in the management science of product development, from many independent sources, indicated that even higher results could be achieved if all the key people involved in the various sequential steps became even more involved in the earlier steps. Researchers and practitioners alike have not yet found the limits on the benefits of the various methods of early involvement.

Why is "Concurrent" the name of choice for the Society? Why not "Parallel," "Integrated," "Interfaced," "Networked," "Collaborative," or "Rapid?" Because none of these words captures the ongoing management science developments that clearly indicate that the key stakeholders of product development activities need to be involved early and, for the most part, continuously. Concurrent is the word that is truest in definition to what the management science has shown produces the best product development results, both for senior management and for product developers.

Why "Product Development" and not "Engineering?" Because the management science shows that to achieve the best results not only must the several engineering disciplines work concurrently but the key cross-functional stakeholders from marketing, product management, purchasing, manufacturing engineering, production, quality, and other functions must be concurrent as well. Perhaps more importantly, many companies and industries that develop products have no engineers. Biologists, nutritionists, chemists, physicists, and other technical disciplines perform the technical roles in cross-functional product development teams. Product Development better captures the reality of the management science and the disciplines of the companies that develop products.

Concurrent Product Development in the News

Two recent media accounts illustrate Concurrent Product Development in action, facilitating multidisciplinary team communication and innovation to speed new products to market...

The fact that the tools and processes described below are not attributed specifically in the source articles as being CPD examples suggests the extent to which CPD is institutionalized in some organizations today.

Virtual aircraft, collocated team

A darkened auditorium, rotating images of a virtual aircraft with the skin removed, viewed on a ten-foot high screen by development team members wearing 3-D glasses. “The image, similar to holography, appears to jump off the screen as if a real aircraft were in the auditorium” (From “3-D Design Software Helps New Aircraft Take Wing,” Los Angeles Times, October 29, 2005).

Dassault Aviation says its new software enabled the French aerospace company to develop its new Falcon 7X business jet in seven months, compared with 16 months for previous business jets. The 3-D software was used not only to design the aircraft, but also to lay out the factory floor plan and create the machine tools, and made it unnecessary to build a mockup or test aircraft

This 3-D process is especially useful in helping development team members determine the best routing of wiring and tubing. How did it work? For a design with 50,000 parts, 300,000 fasteners, and 15 miles of wiring, Dassault says there were only 100 problems when the aircraft was built.



(Continued on page 19)

CPD In The News

(Continued from page 18)

Traditional brainstorming, modern approach



Innovation labs emphasizing collocation of development team members are helping companies like Motorola, Mattel, Boeing, and the Mayo Clinic break down walls between engineering, research, marketing, and other functions. To hustle its Razr ultralight cell phone to market, Motorola, for example, moved designers, marketers, and others to a collocated open space with waist high cubicles, even for senior managers.

At Boeing 3000 engineers, finance and program managers from scattered locations were collocated adjacent to the 737 assembly floor, where you can “feel and hear the noises in the factory and can look out your window and see the wing tips going down that line,” according to Larry Loftis, director of manufacturing. (“Mosh Pits of Creativity— Innovation labs are sparking teamwork and breakthrough products,” *Business Week*, November 7, 2005).

In the same article, Tom Kelley, general manager of IDEO offers **four best practices for building a successful innovation lab**:

- Establish a clearly marked, dedicated place for innovation, conveniently located to staffers’ work places.
- Provide lots of space for white boards and visuals, places where team members can sketch and communicate ideas.
- Apply brainstorming practices that facilitate free thinking; go for quantity, avoid being judgmental.
- Stock the innovation lab liberally with necessities for creativity and prototyping such as sticky notes, fat felt tips, X-Acto knives, and prototyping kits.

The BW article reminds managers they need to get behind winning ideas quickly and not let them perish. “**For big companies that are stuck in creative ruts, modern innovation labs can be places where something magical gets started.**”

Perhaps the most encouraging aspect of the above examples is that the tools and best practices of concurrent product development continue to be applied and refined in forward thinking organizations, and are being “discovered” and reported in today’s news media.