What is the best way to develop software? Continuing the conversation about agility and plan-driven methods

Stan Rifkin
MASTER SYSTEMS INC.
2604B El Camino Real 244
Carlsbad, California 92008 USA
+1 760 729 3388 sr @ Master-Systems.com

Special for the San Diego Software Process Improvement Network. Version 0.2

Outline

- My biases + Take away
- Conversation as of a little over one year ago
- Updates
  - Feedback on Boehm & Turner book
  - A little bit more experience
  - Some empirical field studies
- What is still unknown
  - CMM(I) compliance
  - Open & free sources software
  - Globally distributed software
  - Systems of systems

My biases

- **“Old school.” Just plain old.**
- **Understand in a deep way the need to do better. Am a Certified Scrum Master.**
- **Seeing growth in BOTH long/big waterfall projects (Future Combat Systems) AND shorter, innovative ones (web systems).**
- **Worked/studied at the Software Engineering Institute, a bastion of tradition. But rebelled!**
- **Manager at heart.**
- **Empirical at heart. Though appreciate a good theory!**

If you can't stay …

- **Maybe this should be called "high ritual" vs. "low ritual."**
- **None of the practices are new, so it's the synergism, connections, observations that are new. Also new (well, tangible): the polemic between process & people.**
- **The emphasis on risk is new, so is value-driving the selection of methods.**
- **One size does not fit all: there are projects that are better suited towards the agile end and others better suited towards the plan-driven side.**
- **There are many kinds of projects where we do not know the best (combination of) methods.**

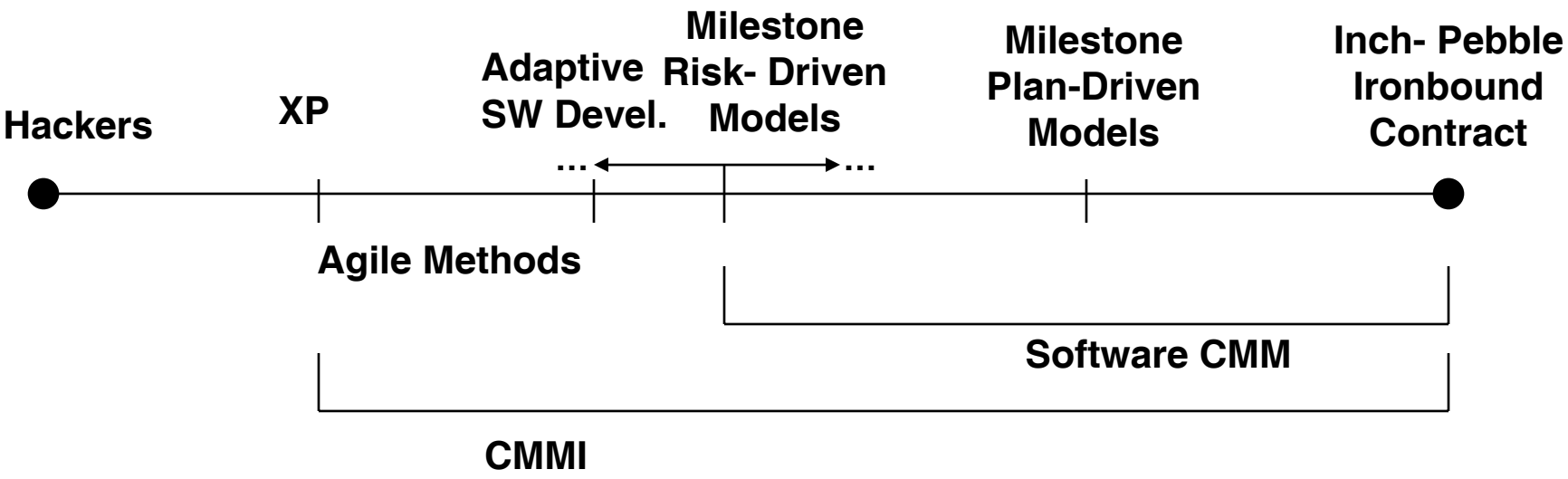
Acknowledgements

- **Barry Boehm, Center for Software Engineering, University of Southern California.**
- **Rich Turner, now of Systems & Software Consortium, consultant to Office of the Secretary of Defense.**
- **Laurie Williams, North Carolina State University, researcher, particularly on pair programming.**
- **Ken Schwaber, co-creator of Scrum.**
- **XPSD – the local group actively interested in agile methods.**
- **Mary Shaw, Carnegie Mellon, for the use of "ritual" to characterize methods. Also, "incantation"!**

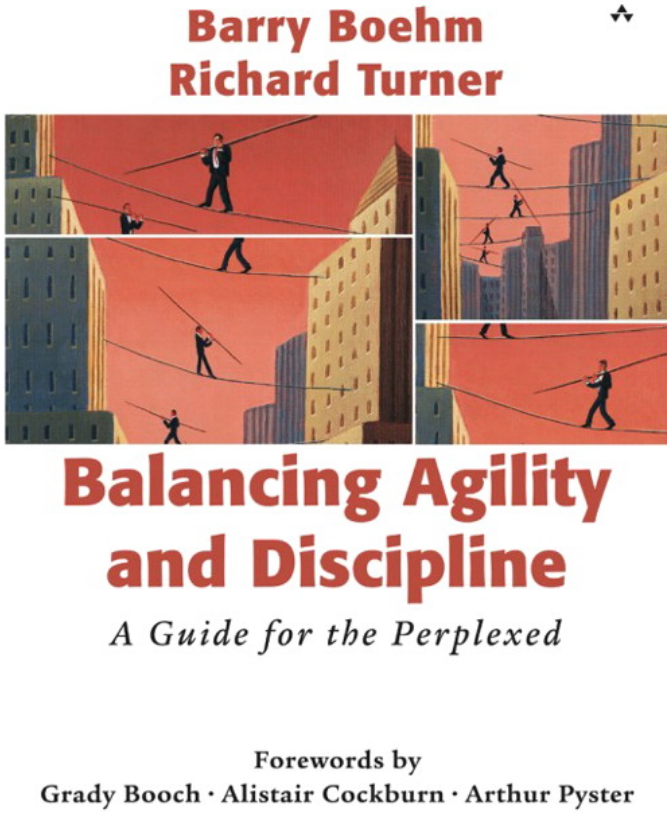
Agile and Plan-Driven Home Grounds

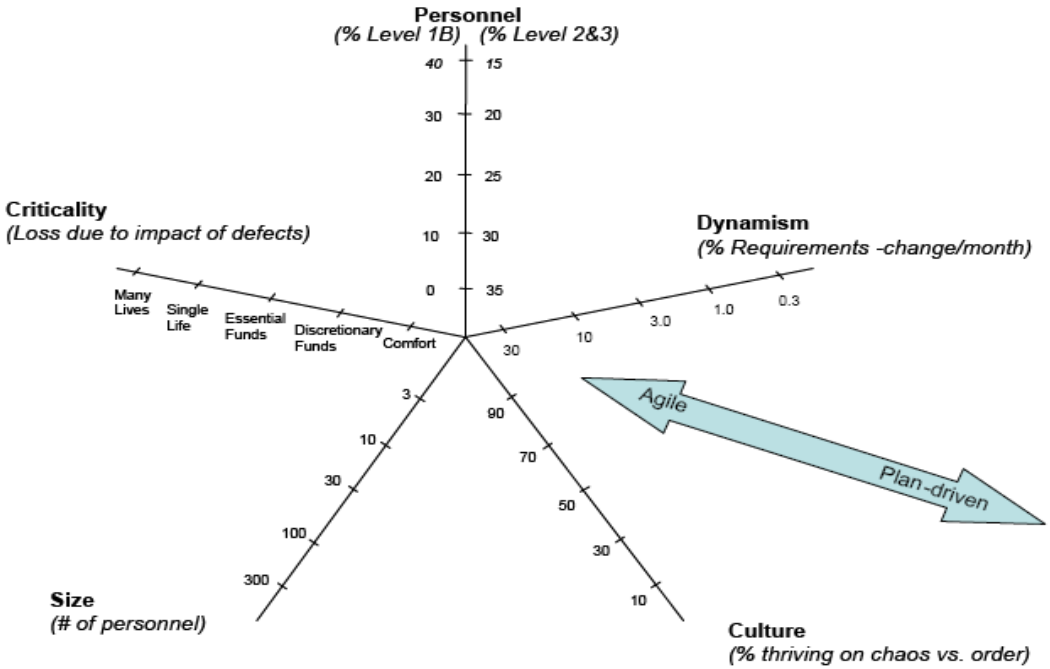
Agile Home Ground	Plan-Driven Home Ground
❖ Agile, knowledgeable, collocated, collaborative developers	❖ Plan-oriented developers; mix of skills
❖ Above plus representative, empowered customers	❖ Mix of customer capability levels
❖ Reliance on tacit interpersonal knowledge	❖ Reliance on explicit documented knowledge
❖ Largely emergent requirements, rapid change	❖ Requirements knowable early; largely stable
❖ Architected for current requirements	❖ Architected for current and foreseeable requirements
❖ Refactoring inexpensive	❖ Refactoring expensive
❖ Smaller teams, products	❖ Larger teams, products
❖ Premium on rapid value	❖ Premium on high-assurance

The Planning Spectrum



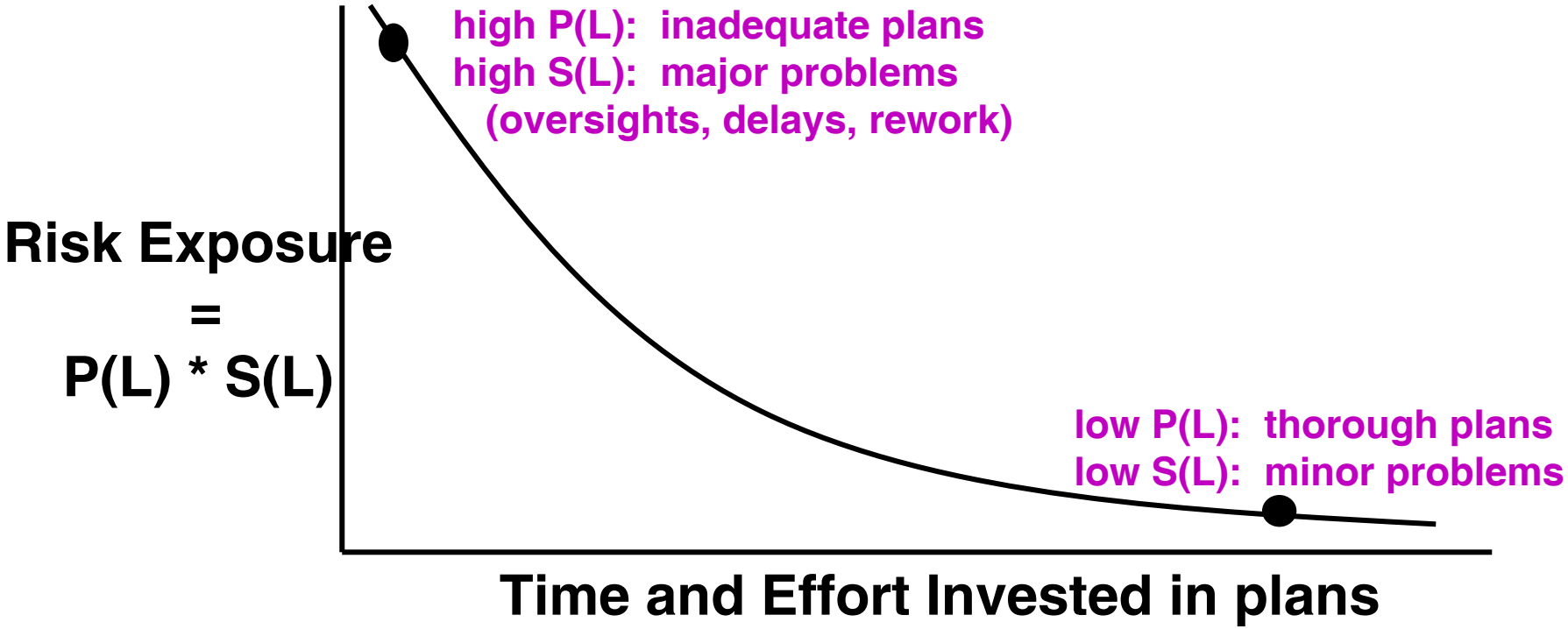
Review (cont.)





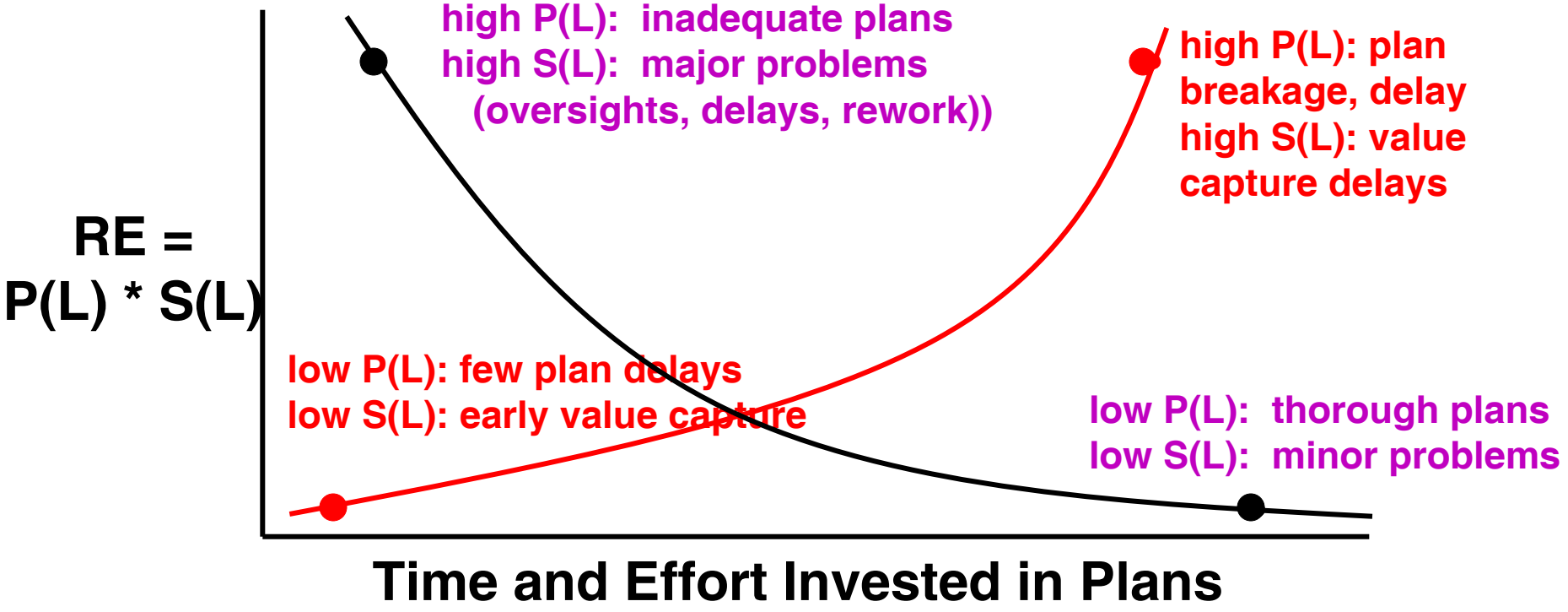
Example RE Profile: Planning Detail

- Loss due to inadequate plans

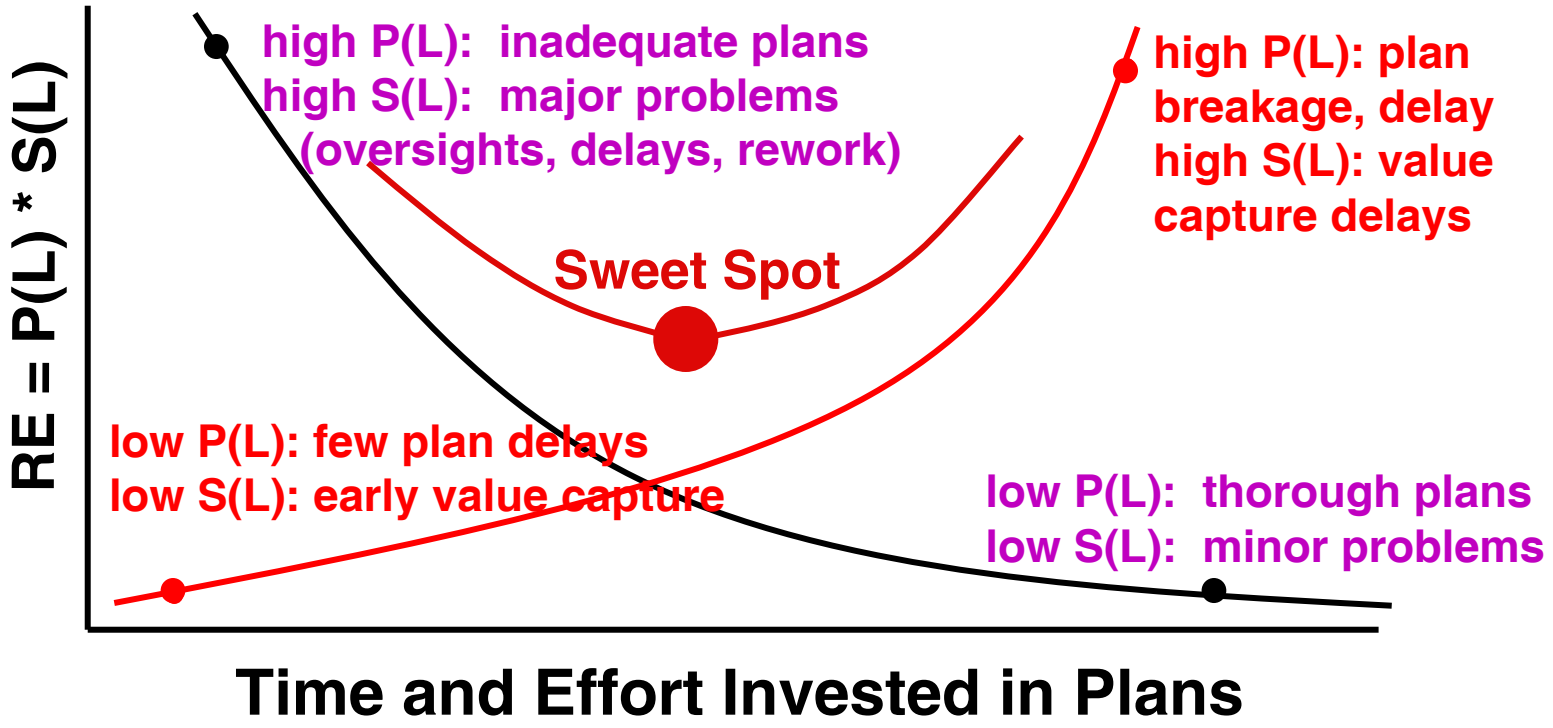


Example RE Profile: Planning Detail

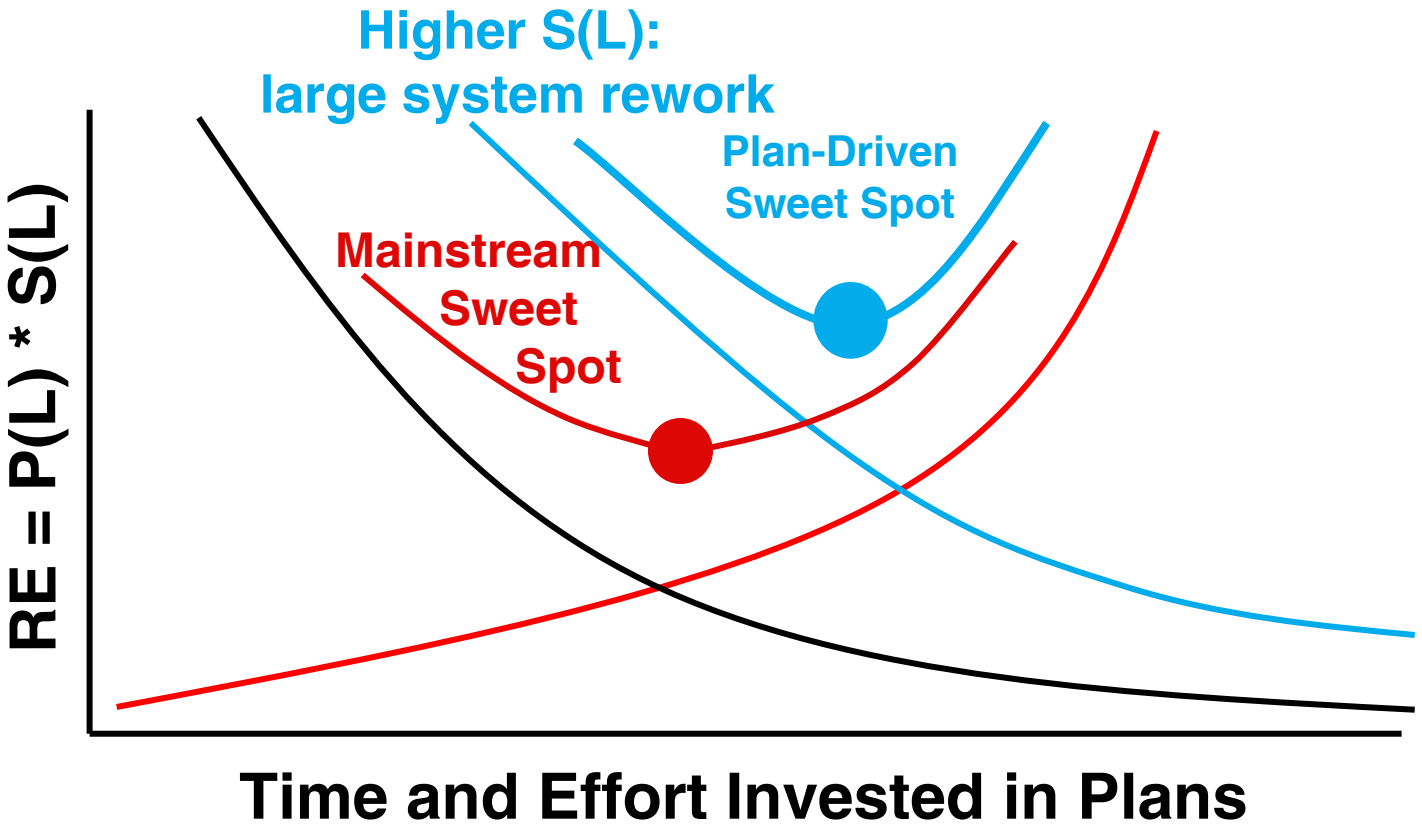
- Loss due to inadequate plans
- Loss due to market share erosion



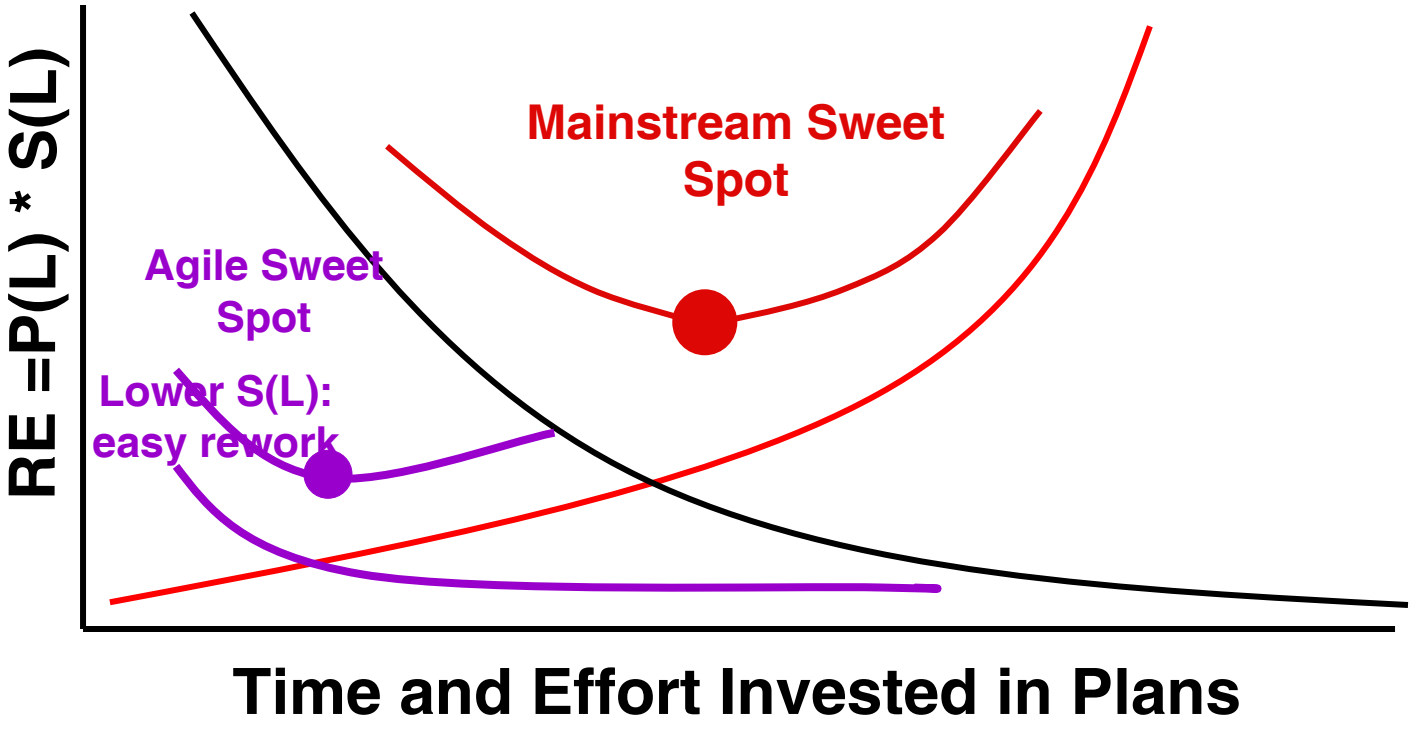
Example RE Profile: Time to Ship
- Sum of Risk Exposures



Comparative RE Profile: Plan-Driven Home Ground



Comparative RE Profile:
Agile Home Ground



What's best way to select methods (redux)?

Table 32: Process Model Decision Table

Objectives, Contraints			Alternatives		Model	Example
Growth Envelope	Understanding of Requirements	Robustness	Available Technology	Architecture Understanding		
Limited			COTS		Buy COTS	Simple Inventory Control
Limited			4GL, Transform		Transform or Evolutionary Development	Small Business - DP Application
Limited	Low	Low		Low	Evolutionary Prototype	Advanced Pattern Recognition
Limited to Large	High	High		High	Waterfall	Rebuild of old system
	Low	High			Risk Reduction followed by Waterfall	Complex Situation Assessment
		High		Low		High-performance Avionics
Limited to Medium	Low	Low-Medium		High	Evolutionary Development	Data Exploitation
Limited to Large			Large Reusable Components	Medium to High	Capabilities-to-Requirements	Electronic Publishing
Very Large		High			Risk Reduction &Waterfall	Air Traffic Control
Medium to Large	Low	Medium	Partial COTS	Low to Medium	Spiral	Software Support Environment

Source: MBASE Guidelines.

Major points

- **Success in selecting the methods depends upon careful characterization of the risks, and therefore …**
- **Success is entirely dependent on selecting projects and methods that fit.**
- **Clearly, one size does not fit all.**

What is agile?

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***Individuals and interactions*** **over processes and tools.**
- ***Working software*** **over comprehensive documentation.**
- ***Customer collaboration*** **over contract negotiation.**
- ***Responding to change*** **over following a plan.**

That is, while there is value in the items on the right, we value the items on the left more.

Source: http://www.agilemanifesto.org/

Agile principles 1-6

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Agile principles 7-12

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Source: http://www.agilemanifesto.org/

4 values (actually XP)

- Simplicity
- Communication
- Feedback
- Courage

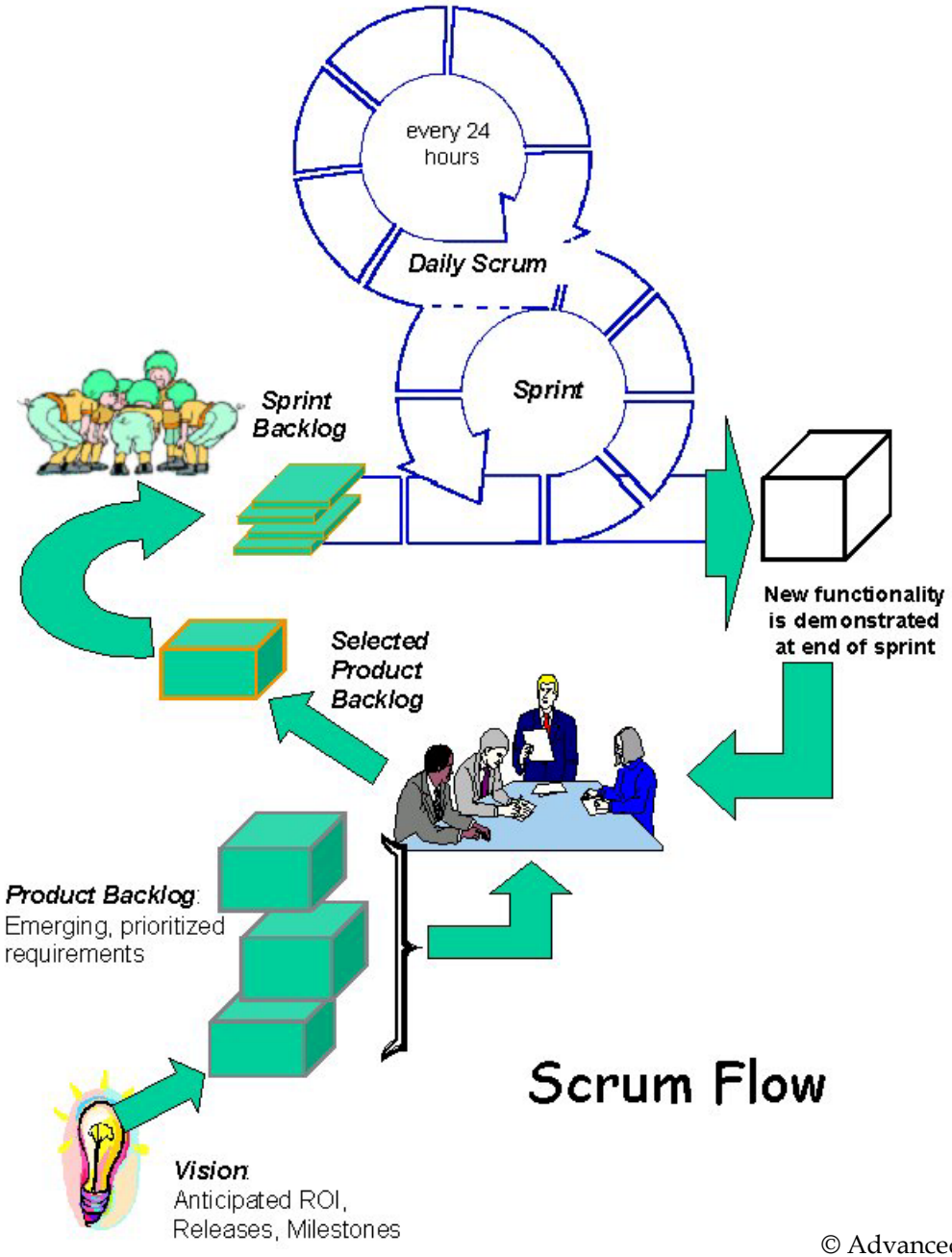
XP2 Primary Practice Summary

XP2 Primary Practice	Sustained/New/ XP1 Name
Sit together	New
Whole team	New
Informative workspace	New
Energized work	40-hour week
Pair programming	Sustained
Stories	Planning game
Weekly cycle	Planning game
Quarterly cycle	Small releases
Slack	New
Ten-minute build	New
Continuous integration	Sustained
Test-first Programming	Testing
Incremental Design	Simple Design Refactoring

XP1 Practice	Disposition
Metaphor	Removed
Collective code ownership	Corollary: Shared code
On-site customer	Corollary: Real customer involvement
Coding standard	Removed

Source: "Case Study Retrospective: Kent Beck's XP Versions 1 and 2," by Laurie Williams, presented at the USC CSE Annual Research Review, March 14, 2005.

Scrum



What's different about Scrum

- **"Potentially implementable or shippable without any significant additional work (friendly first use)"**
- **No project managers: the team is self-directing.**
- **Chickens and pigs. Only pigs can commit.**
- **Does not perform traditional project management. No history to speak of.**
- **Uses a "do a little, then adjust" method.**
- **Can implement one project at a time.**
- **These days its advocates say it's a method by which an organization is transformed.**

Agile methods

- **Programming paradigms**
  - **eXtreme programming**
  - **Feature driven development**
  - **Crystal**
  - **DSDM**
  - **…**
- **Project management paradigms**
  - **Scrum**

- **The programming methods are independent of the project management methods => “plug and play.”**

What about hybrids?

- **One finds them in practice.**
  - **What about Rational Unified Process** (Ron Norman in October 2005) **& Team Software Process?**
- **What makes XP, Scrum, and others work?**
  - **Easily implemented because of bite-size pieces.**
  - **Takes good practices and (appropriately) exaggerates them.**
  - **Answers the call of frustrated developers and their clients. Something new.**

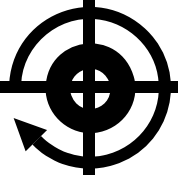
OODA (context-adaptive) loop

Observe

objectives, constraints, alternatives; usage, competition, technology, marketplace

Orient

with respect to stakeholders priorities, feasibility, risks; perform business case/mission analysis; create prototypes, models, simulations



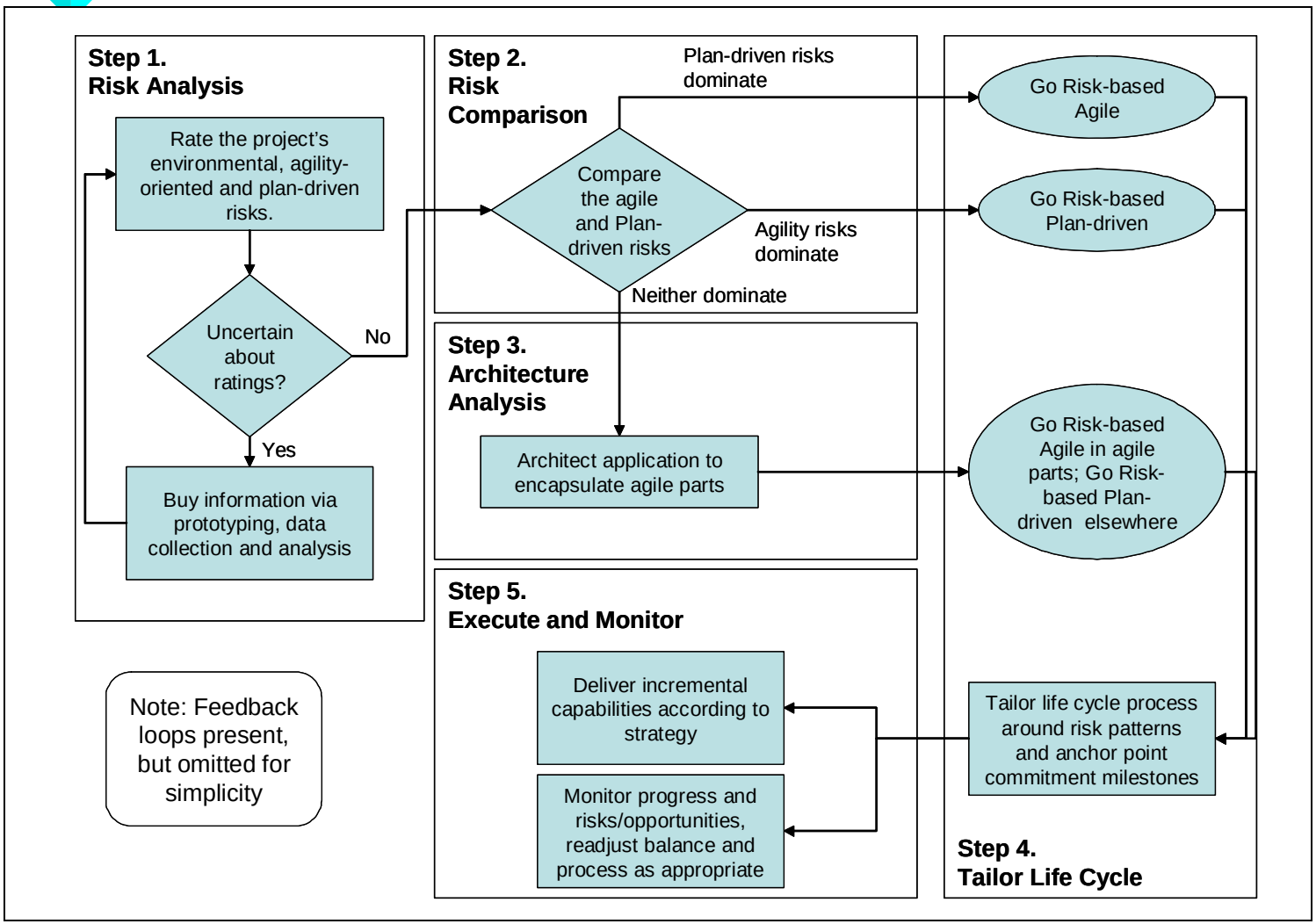
Act

on plans, specifications; keep development stabilized; prepare for next cycle

Decide

on next-cycle capabilities, architecture upgrades, plans; stabilize specifications, COTS upgrades; document development, integration, V&V risks; reassess feasibility (go/no go)

A decision flow for constructing a hybrid



Some concerns about agile

- **Remember, it's not a specific method; there are many methods to choose among.**
- **The list of concerns ebbs & flows with experience and competing ideas.**

Watts Humphrey on XP & TSP

- **Advantages**

1. Emphasis on customer involvement: A major help to projects where it can be applied.
2. Emphasis on teamwork and communication: As with the TSP, this is very important in improving the performance of just about every software team.
3. Programmer estimates before committing to a schedule: This helps to establish rational plans and schedules and to get the programmers personally committed to their schedules-a major advantage of XP and TSP.
4. Emphasis on responsibility for quality: Unless programmers strive to produce quality products, they probably won't.
5. Continuous measurement: Since software development is a people-intensive process, the principal measures concern people. It is therefore important to involve the programmers in measuring their own work.
6. Incremental development: Consistent with most modern development methods.
7. Simple design: Though obvious, worth stressing at every opportunity.
8. Frequent redesign, or refactoring: A good idea but could be troublesome with any but the smallest projects.
9. Having engineers manage functional content: Should help control function creep.
10. Frequent, extensive testing: Cannot be overemphasized.
11. Continuous reviews: A very important practice that can greatly improve any programming team's performance (few programmers do reviews at all, let alone continuous reviews).

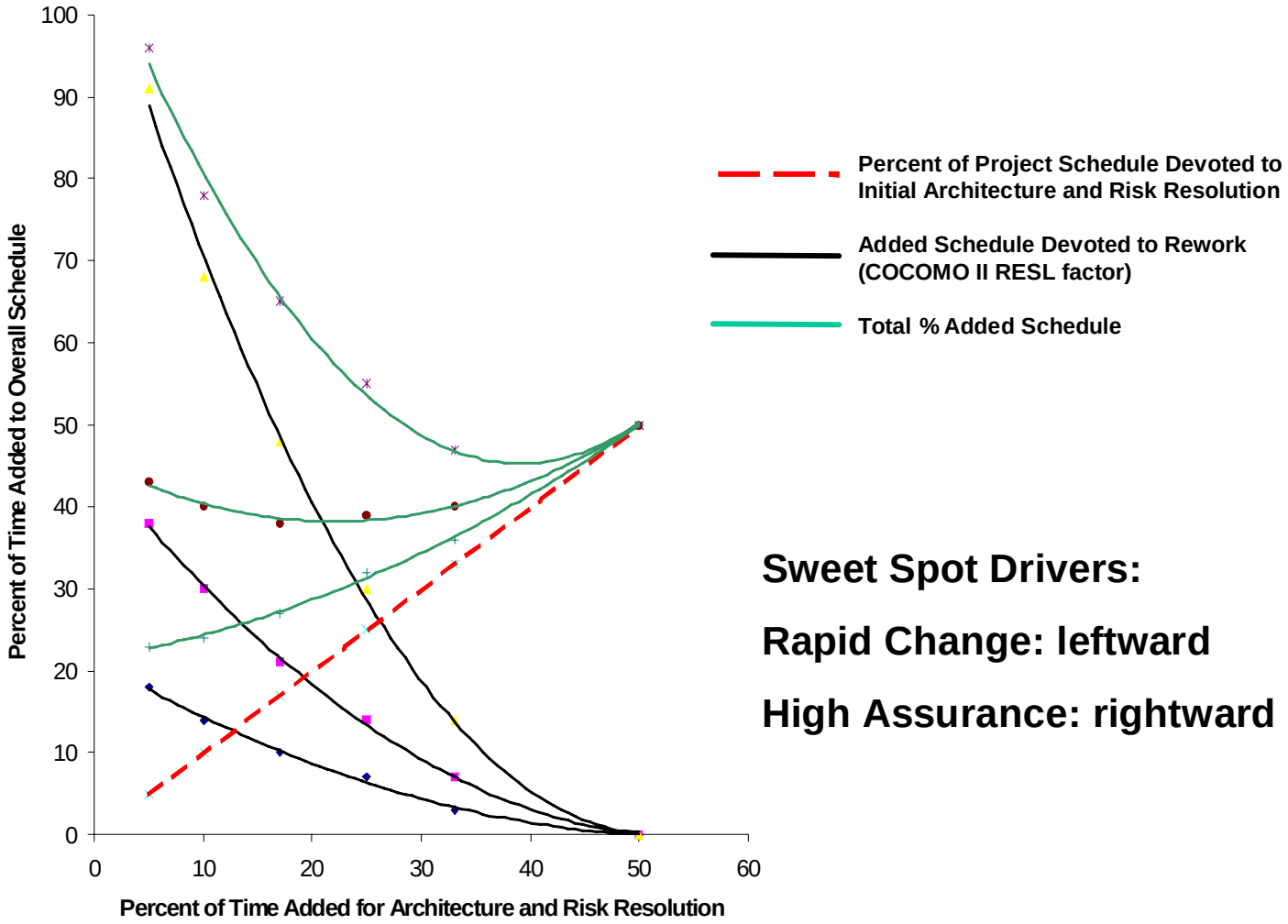
Source: http://www.computer.org/software/dynabook/HumphreyCom.htm

Humphrey on XP & TSP (cont.)

- **Disadvantages**

1. Code-centered rather than design-centered development: Although the lack of XP design practices might not be serious for small programs, it can be disastrous when programs are larger than a few thousand lines of code or when the work involves more than a few people.
2. Lack of design documentation: Limits XP to small programs and makes it difficult to take advantage of reuse opportunities.
3. Producing readable code (XP's way to document a design) has been a largely unmet objective for the last 40-plus years. Furthermore, using source code to document large systems is impractical because the listings often contain thousands of pages.
4. Lack of a structured review process: When engineers review their programs on the screen, they find about 10-25% of the defects. Even with pair programming, unstructured online reviews would still yield only 20-40%. With PSP's and TSP's structured review process, most engineers achieve personal review yields of 60-80%, resulting in high-quality programs and sharply reducing test time.
5. Quality through testing: A development process that relies heavily on testing is unlikely to produce quality products. The lack of an orderly design process and the use of unstructured reviews mean that extensive and time-consuming testing would still be needed, at least for any but the smallest programs.
6. Lack of a quality plan: We have found with the TSP that quality planning helps properly trained teams produce high-quality products, and it reduces test time by as much as 90%. XP does not explicitly plan, measure, or manage program quality.
7. Data gathering and use: We have found with the TSP that, unless the data are precisely defined, consistently gathered, and regularly checked, they will not be accurate or useful. The XP method provides essentially no data-gathering guidance.
8. Limited to a narrow segment of software work: Since many projects start as small efforts and then grow far beyond their original scope, XP's applicability to small teams and only certain kinds of management and customer environments could be a serious problem.
9. Methods are only briefly described: While some programmers are willing to work out process details for themselves, most engineers will not. Thus, when engineering methods are only generally described, practitioners will usually adopt the parts they like and ignore the rest. Kent Beck notes that, when the XP method fails in practice, this is usually the cause.
10. Obtaining management support: The biggest single problem in introducing any new software method is obtaining management support. The XP calls for a family of new management methods but does not provide the management training and guidance needed for these methods to be accepted and effectively practiced.
11. Lack of transition support: Transitioning any new process or method into general use is a large and challenging task. Successful transition of any technology requires considerable resources, a long-term support program, and a measurement and analysis effort to gather and report results. I am not aware of such support for the XP.

Cost of architecture



Sweet Spot Drivers:

Rapid Change: leftward

High Assurance: rightward

Effects of Test Driven Design

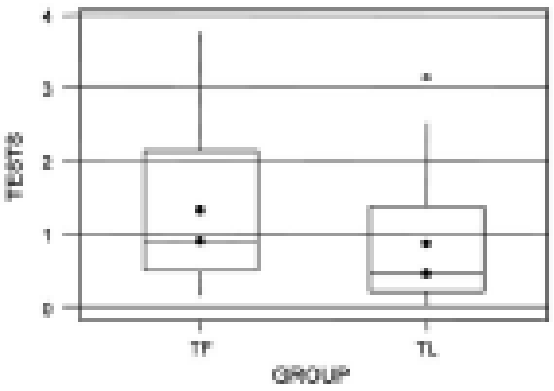
TABLE 1
Summary of Empirical Studies of TDD

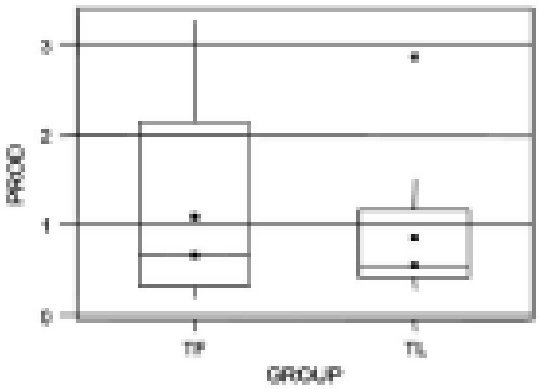
Authors	Study Type	Subjects	Study Scope	Test-First focus?	Tests required?	Interfaces provided?	Incr. dev.?	Quality	Prod.
Müller and Tichy [8]	Case Study	Students	XP	No	?	?	N/A	?	?
Müller and Hagner [9]	Cont. Exp.	Students	TDD	Yes	Yes	Yes	No	No diff.	No diff.
Maximilien, Williams and Vouk [10]	Case Study	Prof.	TDD	No	No	No	No	TDD is better	No diff.
Edwards [11]	Case Study	Students	TDD	No	?	?	N/A	TDD is better	Unknown
George and Williams	Cont. Exp.	Prof.	TDD	No	No	No	No	TDD is better	TDD is worse
This Study	Cont. Exp.	Students	TDD	Yes	Yes	No	Yes	See Sect. 4	See Sect. 4

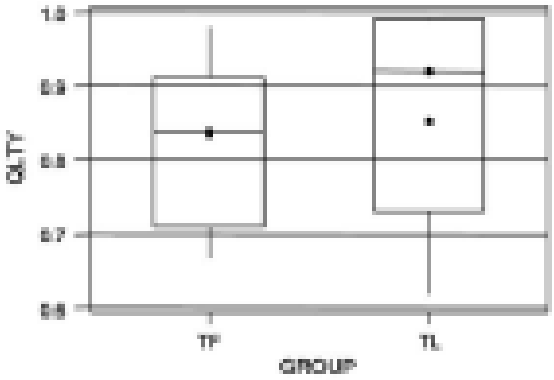
Abbreviations -- Cont. Exp.: Controlled Experiment; Incr. dev.: Incremental Development; Prof.: Professionals; Prod.: Productivity; ?: Unknown/Unspecified; N/A: not applicable.

Source: On the Effectiveness of the Test-First Approach to Programming, Hakan Erdogmus, Maurizio Morisio & Marco Torchiano. IEEE *TRANSACTIONS ON SOFTWARE ENGINEERING*, VOL. 31, NO. 3, MARCH 2005, 226-237 (25 refs).

Test First vs. Test Last







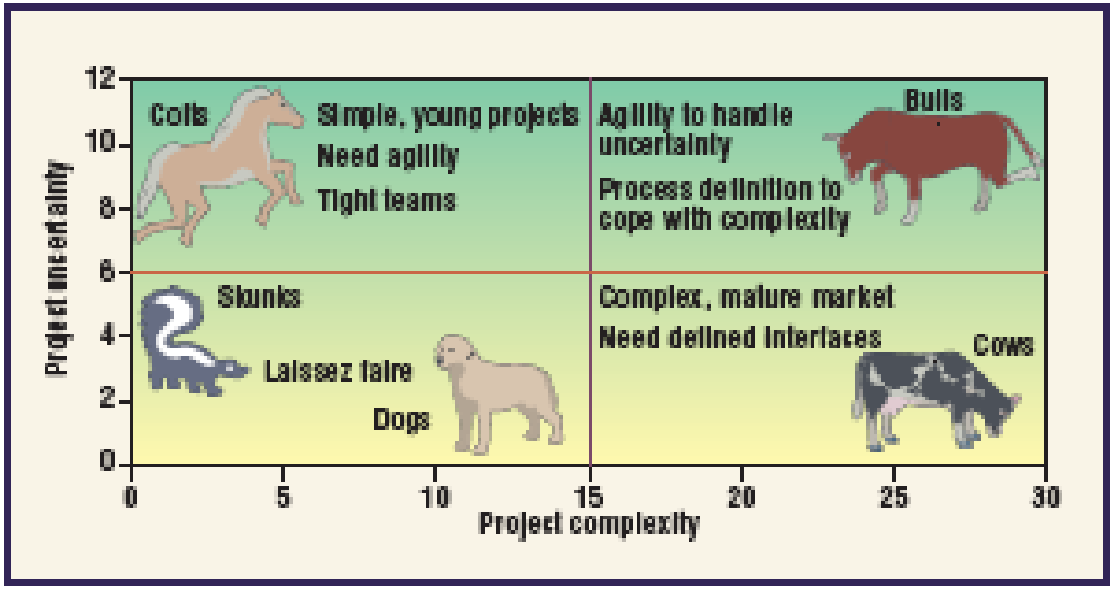
Impact of pair programming on product quality

Metric	Existing empirical body of evidence	Findings of the present study
PP effort percent	No evidence	New: Effort spent on PP is highest in the beginning of a project and in the final iteration
Productivity	Pairs are more productive than solos; PP productivity gradually increases	Not supported: neither programming style has consistently higher productivity
Rationale for PP	PP is most useful for complex tasks and training/ learning	Supported: PP is most useful for learning, and complex tasks
Density of coding standard deviations	PP produces code, which has higher adherence to coding standards	Contradicted: PP results in lower adherence to coding standards
Comment ratio	Pairs produce more readable code; Pair code has lower comment ratio (one study)	Supported: Code produced by pair programming has higher comment ratio than solo code
Relative defect density	Pairs produce code with fewer defects	Not supported: Conflicting results

Source: A Multiple Case Study on the Impact of Pair Programming on Product Quality, Hanna Hulkko & Pekka Abrahamsson, Proceedings of the 27th International Conf on Software Engineering, 2005, pp. 495-504 (34 refs).

Concerns about plan-driven

- Strict (e.g., legalistic) adherence to CMM(I).
- Project management self-fulfilling prophesies.
- Corporate and government acquisition styles.
- Ill-suited to poorly-specified/-understood and/or changing requirements.



Source: Context-adaptive agility: Managing complexity and uncertainty. Todd Little. *IEEE Software,* 22(3), 28-35 (2005).

Areas of greatest methods uncertainty

- **Free and open source software.**
  - **Geographically distributed development.**
- **Systems of (software-intensive) systems: lots of COTS. It's not programming, it's selection, gluing/assembling & tailoring.**
- **CMMI compliance, esp. for small and medium size software development organizations (not projects).**
  - **Some say that agile is CMMI compliant**
    (e.g., Mark Paulk says Scrum is, and see XP CMM.ppt by John Arrizza (cppgent0) on http://groups.yahoo.com/group/xpsandiego/files/)