Learning From Agile Software Development – Part One

Alistair Cockburn
Humans and Technology

This two-part article compares agile, plan-driven, and cost-sensitive software development approaches based on a set of project organization principles, extracting from them ideas for pulling agile techniques into cost- and plan-driven projects. Part one describes how agile and plan-driven teams make different trade-offs of money for information or for flexibility, and presents the first seven of 10 principles for tuning a project to meet various priorities, including cost, correctness, predictability, speed, and agility. Part two, which will run in the November issue of CrossTalk*, will present the last three principles, then pull the material together for actions that plan-driven and cost-sensitive project teams can use to improve their strategies and hedge against surprises.*

Being *agile* is a declaration of prioritizing for project maneuverability with respect to shifting requirements, shifting technology, and a shifting understanding of the situation. Other priorities that might override agility include predictability, cost, schedule, process-accreditation, or use of specific tools.

Most managers run a portfolio of projects having a mix of those priorities. They need to prioritize agility, predictability, and cost sensitivity in varying amounts and therefore need to mix strategies. This article focuses on borrowing ideas from the agile suite to fit the needs of plan-driven and cost-sensitive programs.

Our industry now has enough information to sensibly discuss such blending. The agile voices have been heard [1, 2, 3, 4, 5, 6, 7], the engineering voices have been heard [8, 9, 10], two articles in this issue [11, 12] illustrate the differences in world view, and some authors have discussed the question of their coexistence and principles underlying successful development strategies [3, 8, 13].

Buy Information or Flexibility

Many project strategies revolve around spending money for either information or flexibility [3, 14].

In a *money-for-information* (MFI) proposition, the team can choose to expend resources now to gain information earlier. If the information is not considered valuable enough, the resources are applied to other work. The question is how much the team is willing to expend in exchange for that information.

In a *money-for-flexibility* (MFF) proposition, the team may opt to expend resources to preserve later flexibility. If the future is quite certain, the resources are better spent pursuing the most probable outcome, or on MFI issues.

Different project strategies are made by deciding which issues are predictable, unpredictable but resolvable, or unresolvable, deciding which of those are MFI or MFF propositions, and how best to allocate resources for each.

Predictable issues can be investigated using breakdown techniques. Such an issue might be creating a schedule for work similar to that successfully performed in the past.

Unpredictable but resolvable issues can be investigated through study techniques such as prototypes and simulators. Such issues include system performance limits. These are also MFI propositions. Agile and plan-driven teams are likely to use similar strategies for these issues as part of basic project risk management.

> *"This is a MFI [money-for-information] situation: It is worth spending a lot of money now to discover ... where those next defects are located."*

Unresolvable issues tend to be sociological, such as which upcoming standard will gain market acceptance, or how long key employees will stay around. These issues cannot be resolved in advance, and so are not MFI propositions, but are MFF propositions. Agile and plan-driven teams are intrinsically likely to use different strategies for these issues. Agile teams will set up to absorb these changes, while plan-driven project teams must, by definition, create plans for them.

Teams will differ on which issues are resolvable, and how much money should be spent in advance on predictable issues. A plan-driven team is more likely to decide that creating the project plan is basically a predictable issue, and that a good MFI strategy is to spend resources early to make those predictions.

In contrast, an agile team might decide that the project plan is fundamentally unresolvable past a very simple approximation. There being no effective MFI strategy, it adopts an MFF approach, making an approximate plan early and allocating resources for regular re-planning over the course of the project.

Both agile and plan-driven developers might agree that the question of system performance under load is an important MFI issue, and so both might agree to spend money early to build a simple system simulator and load generator to stress-test the design.

They are likely to spend money differently on design issues. The plan-driven team, viewing it as a sensible MFI proposition, will spend money early to reduce uncertainty about the future of the design. Agile teams are more likely to view design as either being inexpensive to change (a poor MFI candidate) or unresolvable (making it an MFF proposition). They are therefore more likely to choose a design early and allocate money to adjust it over time. This difference on design issues is fundamental, since the two groups view the matter from different decision arenas.

Ten Principles

The following 10 principles have shown themselves useful in setting up and running projects. Most of these are known in the literature [3, 4, 8, 21]. My phrasing of them may be slightly different.

1. Different projects need different methodology trade-offs.
2. A little methodology does a lot of good; after that, weight is costly.
3. Larger teams need more communication elements.
4. Projects dealing with greater potential damage need more validation elements.
5. Formality, process, and documentation are not substitutes for discipline, skill, and understanding.

6. Interactive, face-to-face communication is the cheapest and fastest channel for exchanging information.
7. Increased communication and feedback reduces the need for intermediate work products.
8. Concurrent and serial development exchange development cost for speed and flexibility.
9. Efficiency is expendable in non-bottleneck activities.
10. Sweet spots speed development.

The first seven principles are discussed in this article, the last three will be addressed in part two.

1. Different Projects Need Different Methodology Trade-offs

This should be obvious, but it seems to need re-stating at frequent intervals [15, 16, 17, 18].

Figure 1, adapted from Boehm and Port [8], shows one particular aspect of these differences. In this figure, the two diminishing curves show the potential damage to a project from not investing enough time and effort in planning. The two rising curves show the potential damage to the project from spending too much time and effort in planning.

The lines crossing on the left indicate a project for which potential damage is relatively low with under-planning, and relatively high with over-planning. Much commercial software, including Web services fall into this category. The lines crossing on the right indicate a project for which potential damage is relatively high with under-planning, and for which much more planning would have to be done before damage would accrue from delays due to planning. Safety-critical software projects fall into this category.

The curves should make it clear that when there is risk associated with taking a slow, deliberate approach to planning, then agile techniques are more appropriate. When there is risk associated with skipping planning or making mistakes with the plan, then a plan-driven approach is more appropriate. The curves illustrate clearly the home territory of each.

Figure 2 shows a different characterization of project differences [3]. The hori-zontal axis captures the number of people needing to be coordinated, rising from one on the left to 1,000 on the right. The idea is that projects need more coordina-tion elements to their methodology as the number of people increases.

The vertical axis captures the potential damage caused by undetected defects in the system, from loss of comfort to loss of life. The idea is that projects need more validation elements as the potential damage increases.

Each box in the grid identifies a set of projects that might plausibly use the same combination of coordination and validation policies. The label in the box indicates the maximum damage and coordination load common to those projects (thus, D40 refers to projects with 20-40 people and potential loss of discretionary monies). Projects landing in different boxes should use different policies.

The different planes capture the idea that projects run to different priorities, some prioritizing for productivity, some for legal liability, and others for cost, predictability, agility, and so on.

Any one methodology is likely to be appropriate for only one of the boxes on one of the planes. Thus, at least 150 or so methodologies are needed (Capers Jones identifies 37,000 project categories [17]). That number is increased by the fact that technology shifts change the methodologies at the same time.

2. A Little Methodology Does a Lot of Good; After That, Weight is Costly

Figure 3 (see page 12) relates three quantities: the weight of the methodology being used, the size of the problem being attacked, and the size of the team. (*Problem size* is a relative term only. The problem size can drop as soon as someone has an insight about the problem. Even though problem size is highly subjective, some problems are clearly harder for a team to handle than others.) This figure illustrates that adding elements to a team's methodology first helps then hinders their progress [3].

The dashed line shows that a small team, using a minimal methodology, can successfully attack a certain size of problem. Adding a few carefully chosen elements to the methodology allows them to work more effectively and attack a larger problem. As they continue to add to the methodology, they increase the bureaucratic load they put on themselves and, being only a small team, start expending more energy in feeding the methodology than solving the problem. The size of the problem they can successfully attack diminishes.

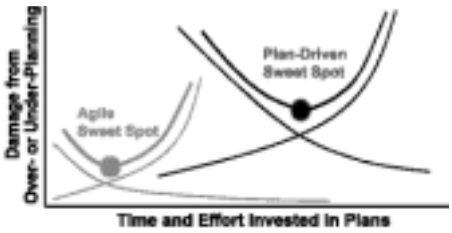


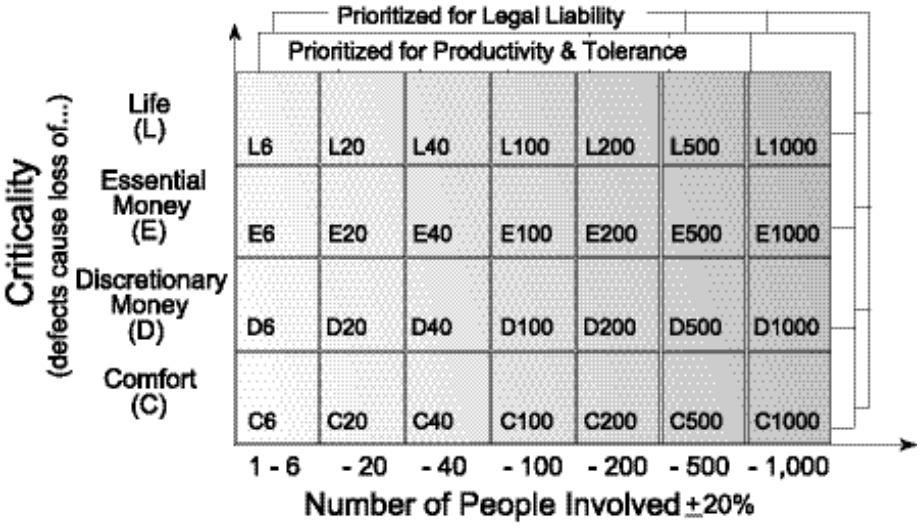
Figure 1: *Balancing Discipline and Flexibility with the Spiral Model and MBASE*

The curve is similar for a large team (the solid line), but not as abrupt. The large team needs more coordination elements to work optimally, and has more people to feed the methodology as it expands. Eventually, even the larger team starts being less productive as the methodology size grows and solves the larger problems less successfully.

3. Larger Teams Need More Communication Elements

Six people in a room can simply talk amongst themselves and write on white boards. If 200 people were to try that, they would get in each other's way, miss tasks, and repeat each other's work. The larger team benefits from coordination. This is the slower rise in the large-team curve in Figure 3. The smaller team needs

Figure 2: *Projects by Communication, Criticality, and Priorities* [3]



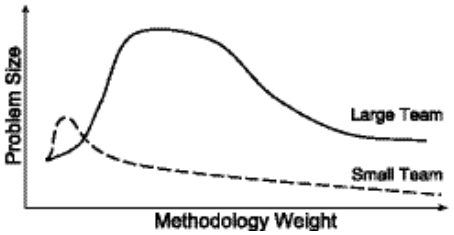


Figure 3: *A Little Methodology Goes a Long Way*

fewer coordination mechanisms and can treat them with less ceremony than can the larger team.

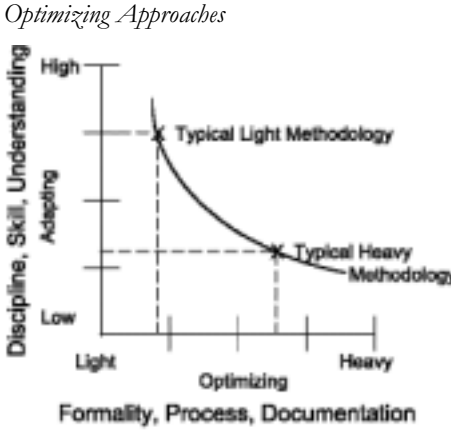
Although this principle should be obvious, many process designers try to find a single set of coordination elements to fit all projects.

4. Projects Dealing with Greater Potential Damage Need More Validation Elements

A team of developers charged with creating a proof-of-concept system does not have to worry about the damage caused by a system malfunction in the same way that a team charged with developing a final production system to be produced in vast quantities does. Atomic power plants, automated weapons systems, even cell phones or automobiles produced in the millions have such economic consequences that it is well worthwhile spending a great deal more time locating and eliminating each additional remaining defect. This is an MFI situation: It is worth spending a lot of money now to discover information about where those next defects are located.

For a system in which remaining defects have lower economic consequences (such as ordering food online from the company cafeteria), it is not worth spending as much money to discover that information. The team will consequently find it appropriate to use fewer and lighter validation techniques on the project

Figure 4: *Differences Between Adapting and Optimizing Approaches*



5. Formality, Process, and Documentation Are Not Substitutes for Discipline, Skill, and Understanding

Highsmith [4] points to the difference between discipline and formality, skill and process, understanding and documentation.

Discipline is an internal quality of behavior; formality is an externally visible result. Many of the best developers are very disciplined in their actions without using formal methods or documents.

Skill is an internal quality of action, typically of a single person, while process is an externally declared agreement, usually between several people. Individuals operating at high levels of skill often cannot say what process they follow. Processes are most useful in coordinating the flow of work between people.

> *"An agile project manager relies on discipline, skill, and understanding, while requiring less formality, process, and documentation."*

Understanding is an internal realization; documentation is external. Only a small part of what people know can be put into external documentation, and that small part takes a lot of time.

Process designers often forget these differences, thinking that enough formality will impart discipline, enough process will impart skill, and enough documentation will impart understanding. An agile project manager relies on discipline, skill, and understanding, while requiring less formality, process, and documentation (Figure 4). This allows the team to move and change directions faster.

6. Interactive, Face-to-Face Communication Is the Cheapest and Fastest Channel for Exchanging Information

Understanding passes from person to person more rapidly when two people are standing next to each other, as when they are discussing at a white board. At that white board, they can use gestures, facial expressions, proximity cues, vocal inflection and timing, cross-modality (aural-visual) timing, and real-time feedback along modalities to discover what each knows, needs to know, and how to convey it [3, 19]. They use the white board as an external-marking device not just to draw, but also to hold some of their discussion points in place so they can refer back to them later.

As characteristics of that situation are removed, the communication effectiveness between the two people drops (Figure 5). On the phone, they lose the entire visual channel and cross-modality timing. With e-mail or instant messaging, they lose vocal inflection, vocal timing, and real-time question and answer. On videotape, they have visuals, but lose the ability to get questions answered. On audiotape, they again lose visuals and cross-modality timing. Finally, communi-cating through documents, they attempt to communicate without the benefit of gestures, vocal inflection and timing, cross-modality timing, proximity cues, or question-and-answer.

This principle suggests that for cost and efficiency improvements, a project team employ personal, face-to-face communication wherever possible. A decade-long study at MIT's Sloan School of Management in the 1970s and a recent research compilation both concluded that physical distance matters a great deal [20, 21].

The cost of imposing distances between people can be seen with a simple calculation. Suppose that a developer earns $2 per minute, and two people working side-by-side on the same problem exchange questions and answers at the rate of 100 questions each per week. Thus, for each minute on average that gets interposed between thinking the question and hearing the answer adds $200 of salary cost to the project per person per week, or about $10,000 per year. For a 10-person project, that one-minute average delay costs the organization $100,000 per year. Two offices being a few meters apart creates a one-minute delay. For offices around the corner or up a flight of stairs, the average delay is more on the order of five minutes ($500,000 per year).

The salary cost is actually the smaller cost. The larger cost is that when two people are more than about half a minute's travel apart, they simply do not ask each other many of those questions. Instead, they guess at the answers. Some percentage of those guesses are wrong, and those mistakes end up as defects in the system that must be found through debugging, external test, integration test, or even through system use.

7. Increased Communication and Feedback Reduces the Need for Intermediate Work Products

Intermediate work products – those not required by the final users of the system or the next team of developers -– tend to have two forms: a) promises as to what will eventually be constructed, and b) intermediate snapshots of the developers' knowledge (design descriptions).

This understanding, as we have already seen, moves faster through interactive than paper-based communication. Increasing the use of interactive communications will never entirely eliminate the need for archivable design documentation, but it can reduce it, particularly during the design and development stages of the project. Eventually, external documentation will be needed when none of the original designers are around, but that does not count as *intermediate* documentation.

Users who regularly get to see the developing system stop needing elaborate promises of what they will be given. This is an MFI issue. If the users are not going to get to see the result for a year or two, then it is worth a lot to create the most accurate promise possible. If on the other hand, the users get to see results every few days or weeks, then a better use of the project's money is to simply build the system and show it to the users.

There is, however, a MFF issue at play here as well since there are diminishing returns on the MFI issue of creating that promise. No amount of care in crafting a detailed promise can capture the unpredictable reaction of the users on seeing the final product in their own environment as they perform their work assignments. The time and money spent on guessing at the users' response to the delivered system would be better allocated to deal with their response on seeing the real system.

Mock-ups, prototypes, and simulations deal with the MFI aspects of the situation. They are an expenditure of resources to discover information sooner. The MFF aspects of the situation are handled through incremental delivery with iterative re-work, allocating resources for the inevitable surprises resulting from real delivery.

Interim Summary

The natural tension between agility-focused, plan-driven, and cost-sensitive project teams is explained in part by their interpretations of what counts as a *money-for-information* proposition, what counts as a *money-for-flexibility* proposition, and how much money to spend on each. We have seen how people with various priorities use those economic strategies differently.

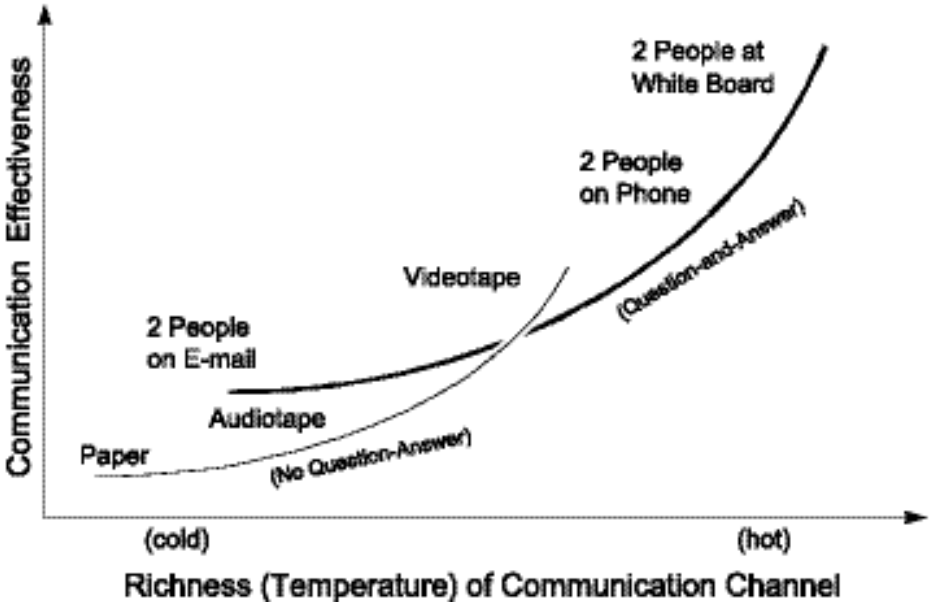


Figure 5: *Increasing Communication Effectiveness from Richer Communication Channels*

It is particularly important, in working with the first seven principles, that each be used to tune a project's running rules, of particular importance is that each project team declares its priorities as well as its communication and validation requirements. With those in place, the team can orient itself to the amount of face-to-face communication it can manage, and the extra methodology weight it should appropriately set in place.

The principles are intended to be used as slider scales. Too much toward each end of the sliding scale brings its own sort of damage.

The second part of this article will present the final three principles, then pull from the collected information to suggest specific actions that leaders of plan-driven and cost-sensitive projects can take to either improve their strategies, or at least hedge their bets against future surprises.◆

References

1. Beck, Kent. eXtreme Programming Explained: Embrace Change. Boston: Addison-Wesley, 1999.
2. Coad, P., E. Lefebvre, and J. De Luca. Java Modeling In Color With UML: Enterprise Components and Process. Upper Saddle River: Prentice Hall, 1999.
3. Cockburn, Alistair. Agile Software Development. Boston: Addison-Wesley, 2001.
4. Highsmith, Jim. Agile Software Development Ecosystems. Boston: Addison-Wesley, 2002.
5. Schwaber, K., and M. Beedle. Agile Software Development with Scrum. Upper Saddle River: Prentice Hall, 2001.
6. Highsmith, Jim, and Alistair Cockburn. "Agile Software Development: The Business of Innovation." IEEE Software 34.9 (2001): 120-122.
7. Cockburn, Alistair, and Jim Highsmith. "Agile Software Development: The People Factor." IEEE Software 34:11 (2001): 131-133.
8. Boehm, Barry, and D. Port. "Balancing Discipline and Flexibility with the Spiral Model and MBASE." CROSSTALK Dec. 2001: 23-30. Available at: <www.stsc.hill.af.mil/crosstalk/2001/dec/boehm.pdf>.
9. Software Engineering Institute. Capability Maturity Model® Integration℠ V1.1 (CMMI℠). Pittsburgh: SEI, 2002. Available at: <www.sei.cmu.edu/cmmi>.
10. Humphrey, Watts. A Discipline for Software Engineering. Boston: Addison-Wesley, 1997.
11. Paulk, Mark C. "Agile Methodologies and Process Discipline." CROSSTALK Oct. 2002: 15-18.
12. Highsmith, Jim. "What Is Agile Software Development?" CROSSTALK Oct. 2002: 4-9.
13. Cockburn, Alistair. "Agile Software Development Joins the Would-Be Crowd." Cutter IT Journal Jan. 2002: 6-12.
14. Sullivan, K., P. Chalasani, S. Jha, and V.

Sazawal. "Software Design as an Investment Activity: A Real Options Perspective," in Real Options and Business Strategy: Applications to Decision Making. L. Trigeorgis, ed. London: Risk Books. Dec. 1999.
15. Mathiassen, L. "Reflective Systems Development." Scandinavian Journal of Information Systems. Vol. 10, No. 1, 2. Gothenburg, Sweden: The IRIS Association, 1998: 67-117.
16. Hohmann, L. Journey of the Software Professional. Upper Saddle River: Prentice-Hall, 1997.
17. Jones, Capers. Software Assessments, Benchmarks, and Best Practices. Boston: Addison-Wesley, 2000.
18. Cockburn, Alistair. "Selecting a Project's Methodology." IEEE Software 17.4 (2000): 64-71.
19. McCarthy, J., and A. Monk. "Channels, Conversation, Cooperation and Relevance: All You Wanted to Know About Communication But Were Afraid to Ask." Collaborative Computing 1.1 (Mar. 1994): 35-61.
20. Allen, T.J. Managing the Flow of Technology. Cambridge, MIT Press, 1977.
21. Olson, G. M., and J. S. Olson. "Distance Matters." Human-Computer Interaction 15 (2001): 139-179.

About the Author

Alistair Cockburn, an internationally recognized expert in object technology, methodology, and project management, is a consulting fellow at Cockburn and Associates. He is author of "Surviving Object-Oriented Projects," "Writing Effective Use Cases," and "Agile Software Development," which have won Jolt Productivity Book Awards. He is one of the original authors of the Agile Software Development Manifesto and founders of the AgileAlliance, and is program director for the Agile Development Conference held in Salt Lake City. Cockburn has more than 20 years experience leading projects in hardware and software development.

1814 Fort Douglas Circle
Salt Lake City, UT 84103
Phone: (801) 582-3162
Fax: (775) 416-6457
E-mail: alistair.cockburn@acm.org

WEB SITES

AgileAlliance

www.agilealliance.org/home

The AgileAlliance is a nonprofit organization dedicated to promoting the concepts of agile software development, and helping organizations adopt those concepts, which are outlined by the Agile Software Development Manifesto and can be found on this Web site. The AgileAlliance was designed to be lightweight, initially consisting of a board of directors, one administrator, and a set of bylaws. Just like agile processes, all work and operations within the AgileAlliance is intended to emerge from subsets of members that self-organize into programs.

Agile Development Conference

http://agiledevelopmentconference.com

The Agile Development Conference is a conference on delivering fit-for-purpose software under shifting conditions, using people as the magic ingredient. A number of techniques, practices, and processes have been identified to do this, and more will be found in the future. This conference will discuss people working together to create software, and the tools, techniques, practices, and issues involved. Come here to learn, or, even better, to name them. This conference has recently been funded and is still being organized. Read the conference vision and structure and the request for participation.

Crystal Methodologies

www.alistair.cockburn.us/crystal

Crystal collects together a self-adapting family of "shrink-to-fit," human-powered software development methodologies based on these understandings:

- Every project needs a slightly different set of policies and conventions, or methodology.
- The workings of the project are sensitive to people issues, and improve as the people issues improve, individuals get better, and their teamwork gets better.
- Better communications and frequent deliveries communication reduce the need for intermediate work products.

This site is a resource for people wanting to understand those ideas, to find more about improving skills and teaming, and to identify some project policies to use as a starting point. This site is set up as a museum of information, with exhibit halls, exhibit rooms, exhibits with notes, and a discussion area.

North Carolina State University

www.csc.ncsu.edu

The North Carolina State University's (NCSU's) Computer Science Department cites strengths in the areas of software systems, communications and performance analysis, theory and algorithms, and computer architecture. Founded in 1967, NCSU's is one of the oldest computer science departments in the country and the only one at a state-assisted Research I University. The university hosts a small workshop, "Agile Software Development Methodologies: Raising the Floor or Lowering the Ceiling" at <http://collaboration .csc.ncsu. edu/agile>.

XBreed

www.xbreed.net/index.html

XBreed is the product of mixing SCRUM, eXtreme Programming (XP) and Alexanderian ideas. Information technology is the result of developing multiple applications and shared components as fast as humanly possible. Combining Scrum and XP was very natural: Scrum provides a solid management framework, while XP provides a basic but complete set of engineering practices. The result is a lean but very effective way to run software projects. In addition, Scrum practiced at the application team level – provided a shared resources team is involved – can lead to re-usability. XBreed is a free method. This Web site includes everything you need to know to run XBreed projects.