Tutorial:
Agile Testing and Evaluation

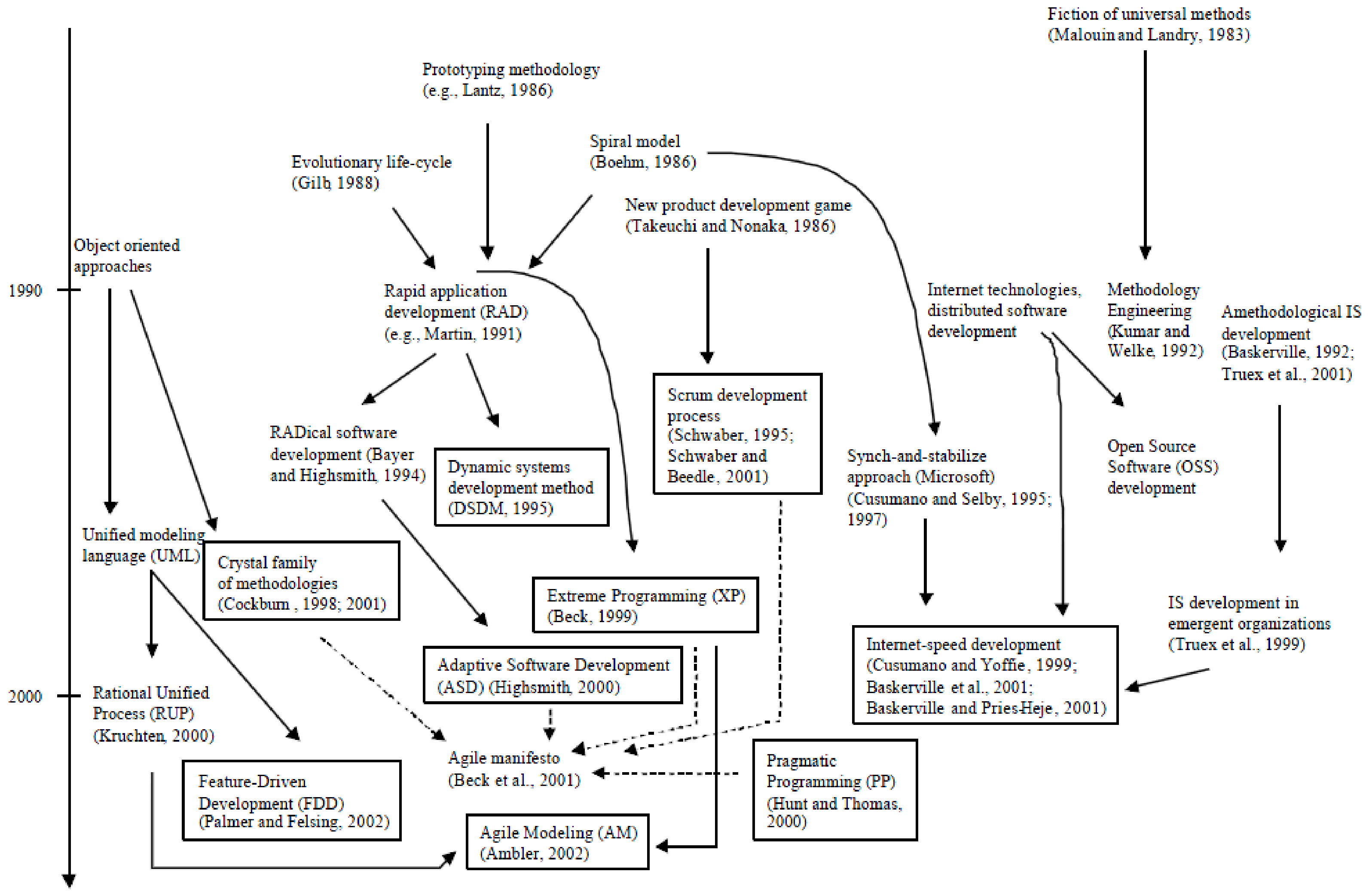
Dr. Robin Poston

University of Memphis

Systems Testing Excellence Program

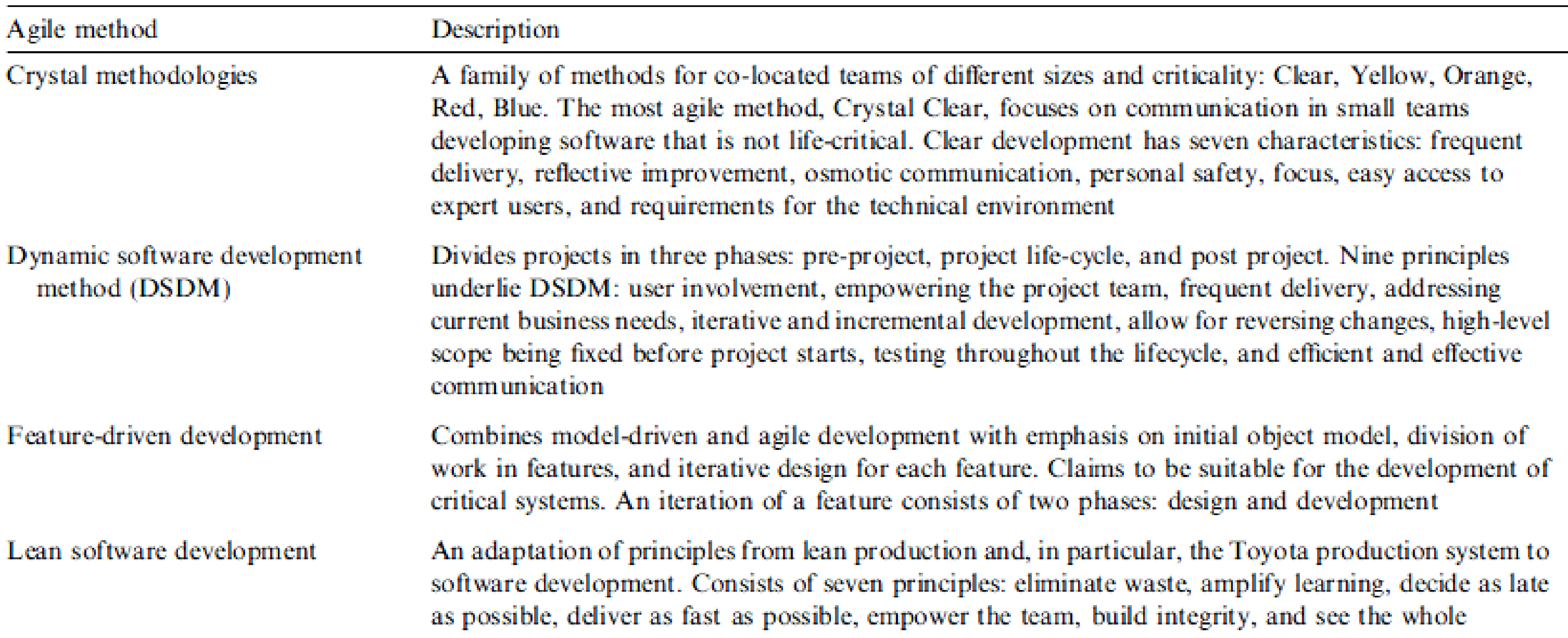
THE UNIVERSITY OF MEMPHIS

Evolutionary Map of Agile Methods


Fiction of universal methods (Malouin and Landry, 1983)
Prototyping methodology (e.g., Lantz, 1986)
Spiral model (Boehm, 1986)
Evolutionary life-cycle (Gilb, 1988)
New product development game (Takeuchi and Nonaka, 1986)
Object oriented approaches
1990
Rapid application development (RAD) (e.g., Martin, 1991)
Internet technologies, distributed software development
Methodology Engineering (Kumar and Welke, 1992)
Amethodological IS development (Baskerville, 1992; Truex et al., 2001)
Scrum development process (Schwaber, 1995; Schwaber and Beedle, 2001)
RADical software development (Bayer and Highsmith, 1994)
Dynamic systems development method (DSDM, 1995)
Synch-and-stabilize approach (Microsoft) (Cusumano and Selby, 1995; 1997)
Open Source Software (OSS) development
Unified modeling language (UML)
Crystal family of methodologies (Cockburn, 1998; 2001)
Extreme Programming (XP) (Beck, 1999)
IS development in emergent organizations (Truex et al., 1999)
Adaptive Software Development (ASD) (Highsmith, 2000)
Internet-speed development (Cusumano and Yoffie, 1999; Baskerville et al., 2001; Baskerville and Pries-Heje, 2001)
2000
Rational Unified Process (RUP) (Kruchten, 2000)
Agile manifesto (Beck et al., 2001)
Pragmatic Programming (PP) (Hunt and Thomas, 2000)
Feature-Driven Development (FDD) (Palmer and Felsing, 2002)
Agile Modeling (AM) (Ambler, 2002)

Source: Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. “New directions on agile methods: a comparative analysis,” In *Software Engineering, 2003. Proceedings. 25th International Conference*, 2003, 244–254.

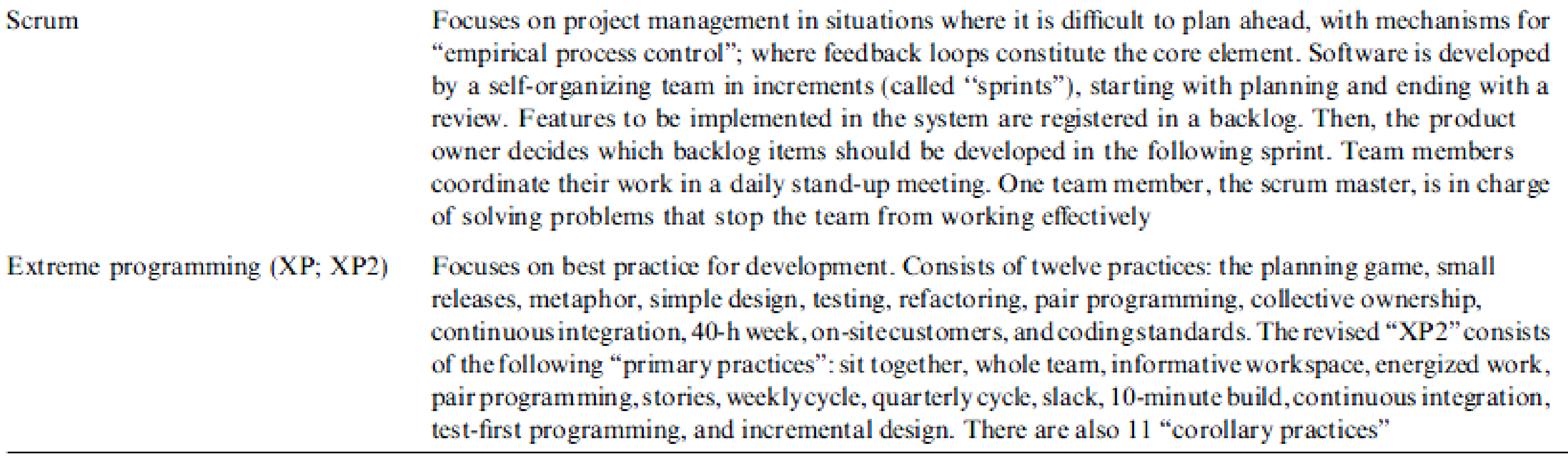
Overview of Agile Methods



Agile method	Description
Crystal methodologies	A family of methods for co-located teams of different sizes and criticality: Clear, Yellow, Orange, Red, Blue. The most agile method, Crystal Clear, focuses on communication in small teams developing software that is not life-critical. Clear development has seven characteristics: frequent delivery, reflective improvement, osmotic communication, personal safety, focus, easy access to expert users, and requirements for the technical environment
Dynamic software development method (DSDM)	Divides projects in three phases: pre-project, project life-cycle, and post project. Nine principles underlie DSDM: user involvement, empowering the project team, frequent delivery, addressing current business needs, iterative and incremental development, allow for reversing changes, high-level scope being fixed before project starts, testing throughout the lifecycle, and efficient and effective communication
Feature-driven development	Combines model-driven and agile development with emphasis on initial object model, division of work in features, and iterative design for each feature. Claims to be suitable for the development of critical systems. An iteration of a feature consists of two phases: design and development
Lean software development	An adaptation of principles from lean production and, in particular, the Toyota production system to software development. Consists of seven principles: eliminate waste, amplify learning, decide as late as possible, deliver as fast as possible, empower the team, build integrity, and see the whole

Dyba, T., & Dingsoyr, T. “Empirical studies of agile software development: A systematic review,” *Information and Software Technology*, 50 (9-10), 2008, 833–859.

Overview of Agile Methods (cont’d.)



Scrum	Focuses on project management in situations where it is difficult to plan ahead, with mechanisms for “empirical process control”; where feedback loops constitute the core element. Software is developed by a self-organizing team in increments (called “sprints”), starting with planning and ending with a review. Features to be implemented in the system are registered in a backlog. Then, the product owner decides which backlog items should be developed in the following sprint. Team members coordinate their work in a daily stand-up meeting. One team member, the scrum master, is in charge of solving problems that stop the team from working effectively
Extreme programming (XP; XP2)	Focuses on best practice for development. Consists of twelve practices: the planning game, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-h week, on-site customers, and coding standards. The revised “XP2” consists of the following “primary practices”: sit together, whole team, informative workspace, energized work, pair programming, stories, weekly cycle, quarterly cycle, slack, 10-minute build, continuous integration, test-first programming, and incremental design. There are also 11 “corollary practices”

Dyba, T., & Dingsoyr, T. “Empirical studies of agile software development: A systematic review,” *Information and Software Technology*, 50 (9-10), 2008, 833–859.

Scrum Key Practices

- Self-directed and self-organizing team
- No external addition of work to an iteration or “sprint,” once chosen
- Daily stand-up meeting with special questions
- Usually time boxed
- Demo to external stakeholders at end of each iteration
- Each iteration, client-driven adaptive planning

Scrum Organization

- 7 or fewer people on a Scrum team
- Common project room
- Can have multiple scrums on a larger project
- “Scrum of scrums” – a daily meeting is also held with a representative of each team

Extreme Programming (XP)

- 4 Values
  - Communication
  - Simplicity
  - Feedback
  - Courage

Extreme Programming (XP)

- 12 Core Practices:
  - Planning Game
    - Release Planning Game sets the scope of the next release by writing features on “story cards” and having the customer choose among them. Iteration Planning Game chooses stories to implement and plans tasks
  - Small, frequent releases
    - Evolutionary delivery: Feedback and emerging information guide development as opposed to a set plan. Iteration is 1-3 weeks in length

Extreme Programming (XP)

- System metaphors
  - E.g., describe a banking system in terms of a manufacturing inventory system
- Simple design
  - Only immediately required components; no duplication
- Testing
  - Write test case before code; continuously rerun test cases

Extreme Programming (XP)

- Frequent refactoring
  - Continuously attempt to simplify the code; also called, “continuous design improvement”
- Pair programming
  - 2 programmers work at one computer rotating typing and observing
- Team code ownership
  - Whole team is responsible for the code

Extreme Programming (XP)

- Continuous integration
  - Newly completed code is integrated into the application code on a separate build machine
- Sustainable pace
  - No overtime should be needed
- Whole team together
  - A customer representative stays with the team all the time
- Coding standards
  - Common coding style

Requirements Elicitation in Agile

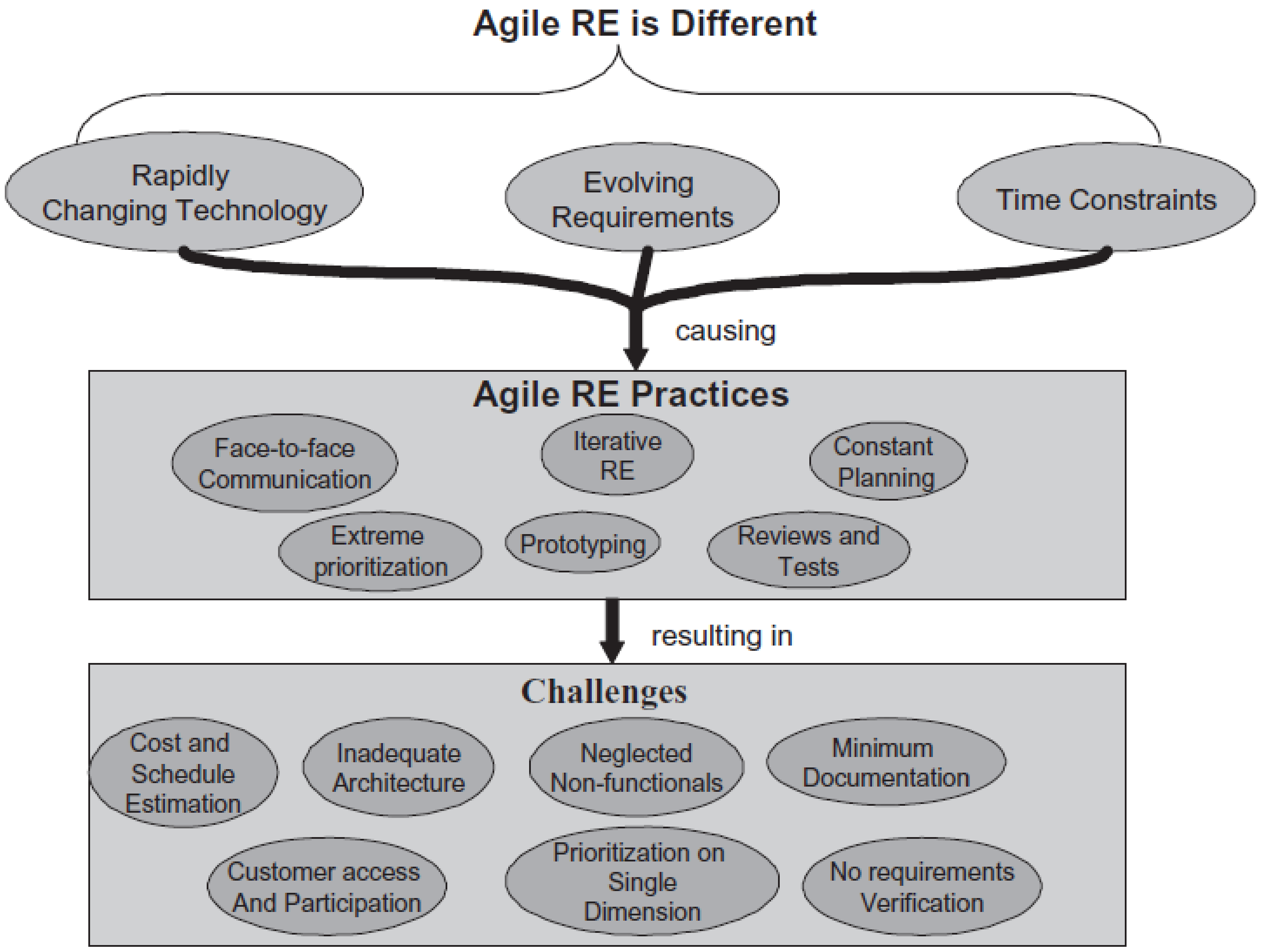


Figure 2. Agile RE practices and challenges.

Combining XP and Scrum

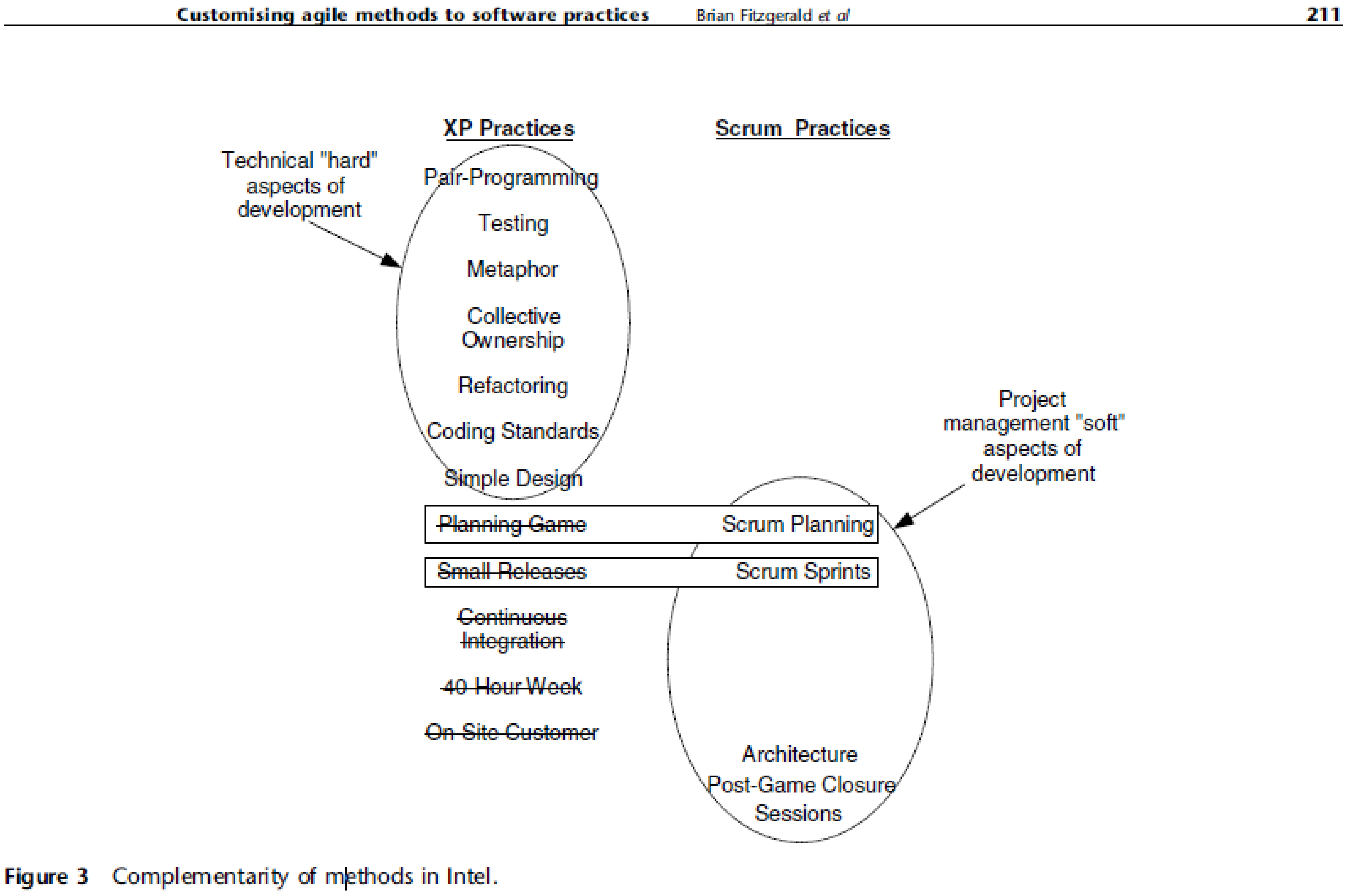


Figure 3 Complementarity of methods in Intel.

Source: Fitzgerald, B, Hartnett, G and Conboy, K. "Customizing agile methods to software practices at Intel Shannon," European Journal of Information Systems, 15 (2), 2006, 197-210.

In-Class Quiz

- Think of a current project you are working on, how can the concepts of agile development be infused to help?
- What would it take for the DoD to run a fully agile development project?
- Are parts of agile useful or should it be an all or nothing adoption?

Agile Teams

- Customer Team
  - Writes “stories” or lists features
  - Provide examples “that will drive coding in business-facing tests”
  - Answer questions, review results

- Developer Team
  - Encouraged to take on multiple roles
  - Can include programmers, system administrators, architects, database administrators, technical writers, security specialists

Comparison of Traditional and Agile Methods

	Traditional	Agile Development
Fundamental Assumption	Systems are fully specifiable, predictable, and are built through meticulous and extensive planning	High-quality adaptive software is developed by small teams using the principles of continuous design and improvement and testing based on rapid feedback and change
Management Style	Command and control	Leadership and collaboration
Knowledge Management	Explicit	Tacit
Communication	Formal	Informal
Development Model	Life-cycle model (waterfall, spiral or some variation)	The evolutionary-delivery model
Desired organizational form/structure	Mechanistic (bureaucratic with high formalization) aimed at large organizations	Organic (flexible and participative encouraging cooperative social action), aimed at small and medium-sized organizations
Quality Control	Heavy planning and strict control; late, heavy testing	Continuous control of requirements, design and solutions. Continuous testing.

Source: Dyba, T., & Dingsoyr, T. "Empirical studies of agile software development: A systematic review," *Information and Software Technology*, 50 (9-10), 2008, 833–859.

Problems with Traditional Software Development

- Takes too long
- By the time it's done the business needs have changed
- Difficult to add/change requirements beyond the requirements stage
- Have to wait too long to see any results
- Uncreative, demeaning
- Iterations are expensive
- Uncompleted projects have nothing to show for the investment

Systems Testing Excellence Program

THE UNIVERSITY OF MEMPHIS

Benefits of Agile Practices

- Enhanced communication/ shared understanding
- Perceived customer control
- Motivated developers
- Better estimates
- Cohesive teams
- Increased quality and project success

Systems Testing Excellence Program

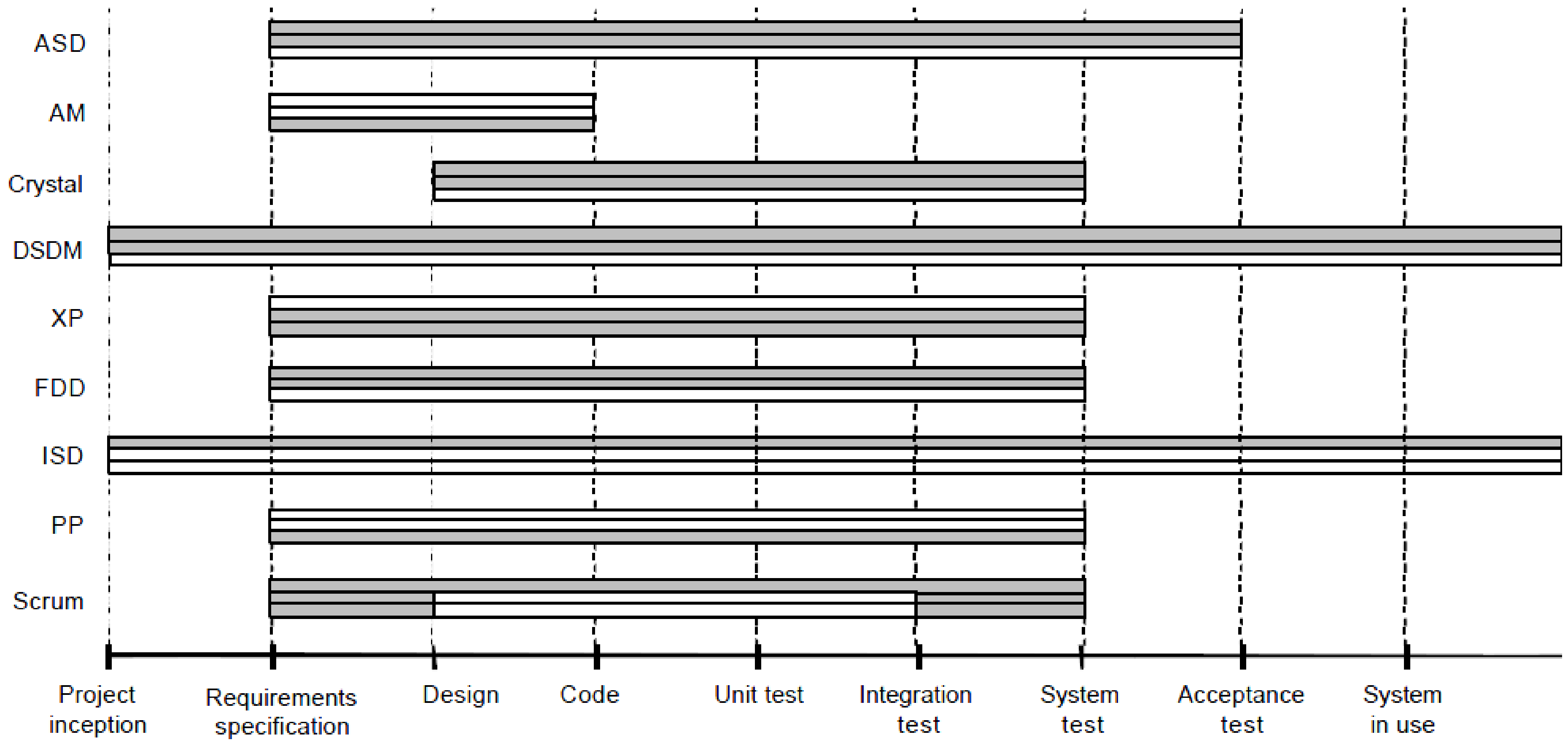
THE UNIVERSITY OF MEMPHIS

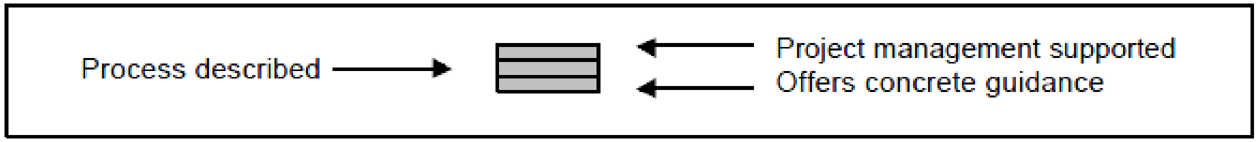
Benefits of Agile Practices

	Customer part of team	Cross funct. team	Self-organizing team	Release often	Collective ownership	Stand up meeting	Retrospective	Kickoff meeting	Prioritized backlog	Iteration backlog	Requirement document	Simple design	Refactoring	Paired programming	Automated tests	Continuous integration	Test-first development	Functional test
Reduce time to market	√	√		√						√		√	√			√	√	
Increase value to market	√			√					√		√							√
Increase quality to market				√	√	√						√	√	√		√	√	√
Increase flexibility	√	√	√		√	√	√		√	√		√	√		√	√		√
Increase visibility				√		√		√	√	√						√		√
Reduce cost		√	√				√		√	√		√	√		√	√		√
Increase product lifetime		√	√		√							√	√	√	√			√

Elssamadisy, A. *Agile Adoption Patterns: A Roadmap to Organizational Success*. Addison-Wesley Professional, 2009.

Fit of Agile Methods Across with SDLC Stages





Source: Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. “New directions on agile methods: a comparative analysis.” *Software Engineering, 2003. Proceedings. 25th International Conference,* 2003, 244–254.

When to Use Agile

- Size Requirements
  - Size of the team matters – normally 7 or fewer
    - Up to 12 person teams - common
    - 12-25– not uncommon
    - 26-100 – few
    - > 100 – isolated

- Personnel Requirements
  - Competency Required? Mixed opinions.
    - Domain expertise, built similar systems, communication skills

- System Context
  - Criticality, reliability, safety depends on testing levels planned

Source: Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R. "Empirical findings in agile methods," *Extreme Programming and Agile Methods—XP/Agile Universe,* 2002, 81–92.

Hybrid Agile Development

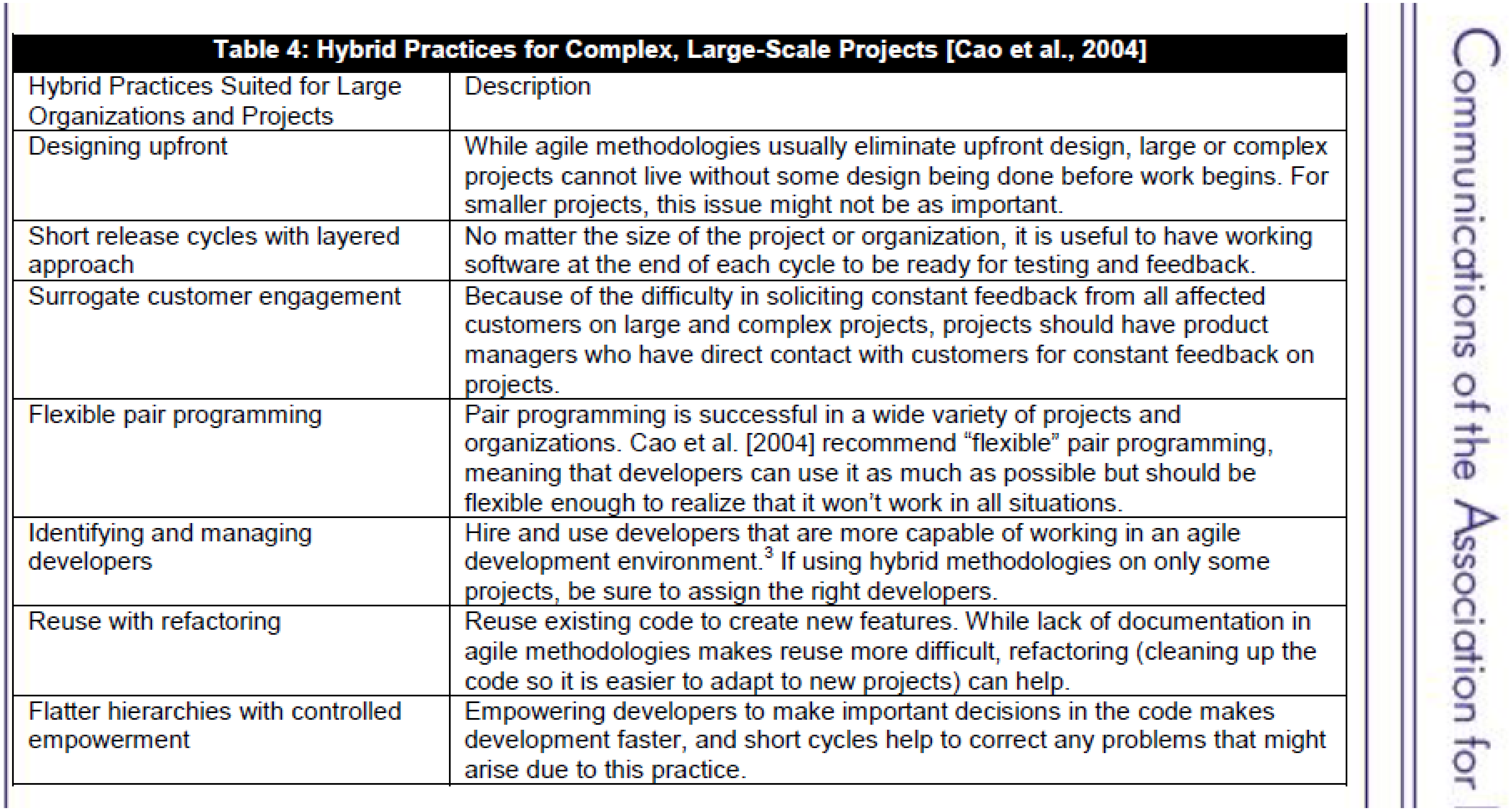


Table 4: Hybrid Practices for Complex, Large-Scale Projects [Cao et al., 2004]

Hybrid Practices Suited for Large Organizations and Projects	Description
Designing upfront	While agile methodologies usually eliminate upfront design, large or complex projects cannot live without some design being done before work begins. For smaller projects, this issue might not be as important.
Short release cycles with layered approach	No matter the size of the project or organization, it is useful to have working software at the end of each cycle to be ready for testing and feedback.
Surrogate customer engagement	Because of the difficulty in soliciting constant feedback from all affected customers on large and complex projects, projects should have product managers who have direct contact with customers for constant feedback on projects.
Flexible pair programming	Pair programming is successful in a wide variety of projects and organizations. Cao et al. [2004] recommend "flexible" pair programming, meaning that developers can use it as much as possible but should be flexible enough to realize that it won't work in all situations.
Identifying and managing developers	Hire and use developers that are more capable of working in an agile development environment.[3] If using hybrid methodologies on only some projects, be sure to assign the right developers.
Reuse with refactoring	Reuse existing code to create new features. While lack of documentation in agile methodologies makes reuse more difficult, refactoring (cleaning up the code so it is easier to adapt to new projects) can help.
Flatter hierarchies with controlled empowerment	Empowering developers to make important decisions in the code makes development faster, and short cycles help to correct any problems that might arise due to this practice.

Communications of the Association for

In-Class Quiz

- What are the trade-offs in following agile? What is given up and what is gained?
- How does speed to market matter and will agile really help?
- How does agile development increase value, quality, visibility, and flexibility? Can it do this for the DoD?

Challenges and Issues in Testing Agile Projects

- Testers may be unfamiliar with the agile development environment
- Testers may not be used to working closely with users or developers – they may be used to working only with other testers and only during well-defined test periods
- Organizational changes – Testers may be removed from traditional testing departments – or not
- Testers may be asked to take on non-testing roles, including acting as developers

Challenges and Issues in Testing Agile Projects

- Testers may not be comfortable with an environment of changing requirements
- This major change of culture may or may not be palatable to experienced testers

Role of a Tester on Agile Projects

- Part of the team
- Works closely with customers to define acceptance tests for each story
- Tests each story as it is complete
- Practices pair testing
- Provides continuous feedback to the team
- Works closely with developers to do performance and other types of testing

Source: Jain, N. at http://www.slideshare.net/nashjain/agile-testing-392928/download

Ten Tips for Agile Testing

1. Integrate the testers in the development teams
2. Use risk-based testing
3. Have testers review unit tests
4. Create a test automation framework (and use it)
5. Display quality metrics in a public location
6. Add a test scrum
7. Implement test retrospectives
8. Add open problems to sprint backlog
9. Testing is still testing
10. Start doing it!

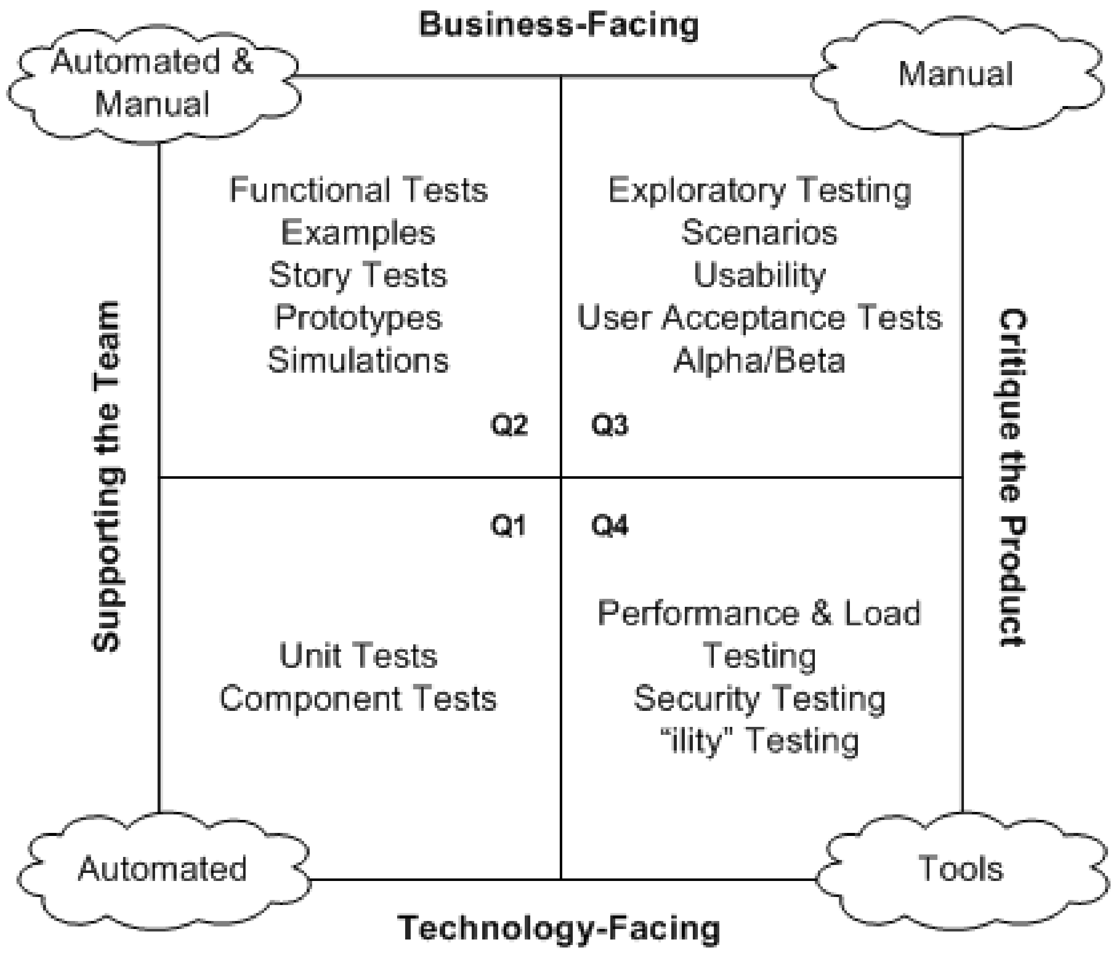
Source: http://softwaredevelopmentisfun.blogspot.com/2008/04/ten-tips-for-agile-testing.html

Challenges in Testing Agile Projects

- Technical
  - Requirements are changing
  - Bringing Testing Forward
  - Moving from manual testing to automation
- Organizational
  - Tester/developer roles are blurred
  - Developer/Test teams might be separate
  - Agile practices require a change in culture

Source: Jain, N. at http://www.slideshare.net/nashjain/agile-testing-392928/download

Agile Testing Quadrants



Source: Crispin, L., and Gregory, J. *Agile testing: a practical guide for testers and agile teams*. Addison-Wesley Professional, 2009.

Managing Agility

- Success Factors
  - Culture – degree of localized control, supportive of negotiation,
  - People – competent team members, trust
  - Communication – rapid communication, co-location, customer feedback
- Warning Signs
  - Daily standups, morale, useless documentation, burn rate, overall iteration schedules
- Architecture
  - Rework/Refactoring, Big Design Up Front (BDUF), Testing is a major issue
- Documentation
  - Live documents

Source: Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R. "Empirical findings in agile methods," *Extreme Programming and Agile Methods—XP/Agile Universe 2002*, 2002, 81–92.

Agile Techniques in the Traditional Development Environment

- Some organizations that don’t want to fully jump into agile can still benefit from agile techniques.
- The use of agile techniques in traditional development can be a lead-in to agile development.
- Some development projects that don’t lend themselves to agile development can benefit from agile techniques.

Agile Techniques in the Traditional Development Environment

- Agile techniques in traditional development:
  - Test-driven development (TDD).
  - Testers engaged earlier in the development cycle.
  - Continuous integration.
  - Customer personnel engaged on a continuous basis.
  - Pair programming.

In-Class Quiz

- How would you go about incorporating parts of agile into a waterfall project?
- What would be gained and what would be lost?
- Have you tried to be agile? How did it go?