If conflict and argumentation are indeed signs of progress, the field of software development must be advancing by leaps and bounds. In the past decade, there has been much debate among software developers over contradictory perspectives of the traditional plan-driven approaches vs. the agile philosophy of software development. Conspicuously absent, however, are the rich perspectives on similar shifts in patterns of thought in other disciplines and the articulation of underlying conceptual assumptions built into today’s software designs. Here, we identify theoretical and conceptual support for the emerging epistemology—methods of searching for and validating basic knowledge—of software practice epitomized in the principles of agile methodologies. Such support is important [4]. Foremost, it enriches our understanding of the agile phenomenon by examining its theoretical roots. It also enables informed use of agile practices in appropriate situations; helps discover reasons for the success of software practice in complex social situations; and facilitates dissemination of agile concepts.

Software development is a complex undertaking beset with many problems, called “wicked problems” by Horst Rittel, an urban planner who pioneered the concept of issue-based information systems to facilitate the formulation and clarification of complex administrative decisions [2]. These problems tend to be unique and difficult to formulate, and solutions evolve continually as the designer gains a greater appreciation of what must be solved. The need to satisfy multiple conflicting viewpoints makes it difficult to devise a test to determine the effectiveness of solutions. “Argumentation,” according to [2], is critical in solving such problems. It would appear that modes of inquiry in software development are making way for learning and innovation.
Solving with Iterative Problem Solving with Single-Loop Learning
Responsiveness
Goal of Problem Solving

Solving with Iterative Problem Solving with Double-Loop Learning
Linear Problem Solving with Single-Loop Learning

Figure 1. Evolutionary shifts in design thinking

**Goal of Problem Solving**

- **Unprincipled**
- **Unpredictable**
- **Unpragmatic**
- **Uninventive**
- **Unwilled**

**Strategy Formulation**

- **Unprincipled**
- **Unpredictable**
- **Unpragmatic**
- **Uninventive**
- **Unwilled**

**Decision Making**

- **Unprincipled**
- **Unpredictable**
- **Unpragmatic**
- **Uninventive**
- **Unwilled**

**Implementation**

- **Unprincipled**
- **Unpredictable**
- **Unpragmatic**
- **Uninventive**
- **Unwilled**

**Evaluation**

- **Unprincipled**
- **Unpredictable**
- **Unpragmatic**
- **Uninventive**
- **Unwilled**

**Learning**

- **Unprincipled**
- **Unpredictable**
- **Unpragmatic**
- **Uninventive**
- **Unwilled**

**Design Metaphor**

- **Unprincipled**
- **Unpredictable**
- **Unpragmatic**
- **Uninventive**
- **Unwilled**

**New Conceptualization of Software Development**

- **Unprincipled**
- **Unpredictable**
- **Unpragmatic**
- **Uninventive**
- **Unwilled**

**New Conceptualization of Softw...
behave technical rationality based on observation and facts, or logical positivism [11]. Emerging practices (such as design thinking) use the assumption that change and uncertainty can be controlled through a high degree of formalization. Proponents of agile methods have discovered inadequacies in formal design that follows systematic procedures dictated by rigid processes. These insights have produced a more inclusive method of inquiry that departs from traditional approaches to software development. This conceptual shift and its resulting debates are hardly exceptional; similar evolutionary shifts in thought and methods of inquiry are also found in architecture and strategic management.

Our discussion here is based on the development of design concepts in architecture, as outlined in [2]. Briefly, Chris Alexander, a professor of architecture at the University of California, Berkeley, pioneered in the 1970s on work on patterns in architectural design that inspired software design patterns, a conceptual breakthrough that revolutionized the way software is built. Design patterns are proven design experiences and best practices for solving similar problems across domains. Alexander’s early model conceived design as an instrumental process dictated by technical rationality [2]. It aimed to identify the optimal means to a predictable end by employing scientific methods used in problem solving, the engineering discipline [13]. This view reduced design to a linear sequence of well-defined steps, with analysis preceding synthesis [2]. Systematic problem decomposition is the key activity in analysis, a process reminiscent of hierarchical top-down functional decomposition. Hence, there is a tension between the organization and its environment [6]. This approach is predicated on the assumption of a foreseeable and unchanging world, which, of course, it never is. Moreover, this approach sought the best means to arrive at an end and thorough analysis and reasoning, concordant with functional/technical rationality. According to Henry Mintzberg, a theorist in management and organizational research, the design, planning, and positioning schools of thought in strategic management reflect this view and tend to be highly prescriptive [7].

Consistent with early approaches to architectural design, strategy formulation through logical thinking and planning proceeded implementation of strategic initiatives. Moreover, strategic management was the exclusive province of top management, with little or no involvement of the people who actually implement the strategy. While these planned-ahead schools historically exploited past experience, they lacked focus on learning through exploration and were inflexible and woefully ill-suited for agile response to rapidly changing environments.

The conceptual appeal and practicality of incremental learning in turbulent and complex organizational environments has led many strategists to reposition formulation closer to implementation [6]. In this view, strategy formulation (as an emergent process) is constantly influenced by learning, which occurs during implementation. An early analysis of this approach, conceptualizes strategy formulation as a craft, likening strategists to craftspeople whose minds work in concert with their hands as they learn and improvise in shaping an artifact [7]. Effective strategists evolve through a learning process involving skills, experience, and insights gained through the dynamic interplay among formulation, implementation, and critical reflection. Thus, strategy is an emergent process in which strategy formulation and implementation are inseparable [7].

Alexander’s early model conceived design as an instru-mental process that acknowledges the existence of environmental uncertainty and complexity is evident in today’s strategic thinking. Much can be learned about design efficacy by examining the patterns of inquiry in various disciplines. The predictive and prescriptive models of software development parallel the maturation of design ideas in architecture and strategic management. The traditional mechanistic worldview is today being challenged by a newer agile perspective, according to [5], and this “emergent worldview,” “adductive” (imagining and preparing for a preferred future state), “hypothesis-dri-ven” (conceptually enacting “what-if” and “if-then” scenarios), “opportunistic” (alerting to evolving prospects), “dialectical” (resolving contradictions), “extension,” and “inquiring and value-based” (questioning assumptions with openness while being sensitive to the values of all stakeholders).

Several characteristics of “design thinking,” according to [5], may be gleaned from the conceptually opposing strands of design in the table. Design thinking is a “constructive synthesis” (or “multiple worldviews”), “adductive” (imagining and preparing for a preferred future state), “hypothesis-driven” (conceptually enacting “what-if” and “if-then” scenarios), “opportunistic” (alerting to evolving prospects), and “extension” (resolving contradictions), and “inquiring and value-based” (questioning assumptions with openness while being sensitive to the values of all stakeholders).

In today’s world of increased complexity software engineers need a broader interpretation of the metaphor of design than is generally accepted.

**NEW CONCEPTUALIZATION OF SOFTWARE DEVELOPMENT**

The trend in management thinking, moving from a deterministic/mechanistic view of problem solving to a dynamic process, characterized by iterative cycles and the active involvement of all stakeholders, is reflected in software development as well. The “emergent metaphor of design” in the table is manifest in the agile methods in today’s emerging software-development thinking. Agile methods are people-centric, recognizing the value of competent people and ideas that bring to software development. In addition, it focuses on providing high customer satisfaction through three principles: quick delivery of quality software; active participation of concerned stakeholders; and creating and
Design process

Traditional View of Design

- Designers:
  - Discrete and sequential steps
  - Linear, hierarchical flow

- Processes:
  - Plan->Build
  - Fix->Test
  - Evaluate->Revise

- Risk:
  - High

Emergent Metaphor of Design

- Designers:
  - Continuous and iterative
  - Interactive, multidisciplinary

- Processes:
  - Plan-Do-Check-Act
  - Reflectively

- Risk:
  - Low

Goal

Traditional View of Design

- Focus:
  - External
  - Customer needs

- Processes:
  - Plan->Build
  - Fix->Test
  - Evaluate->Revise

- Risk:
  - High

Emergent Metaphor of Design

- Focus:
  - Internal
  - Team needs

- Processes:
  - Plan-Do-Check-Act
  - Reflectively

- Risk:
  - Low

Key characteristics

Traditional View of Design

- Focus:
  - External
  - Customer needs

- Processes:
  - Plan->Build
  - Fix->Test
  - Evaluate->Revise

- Risk:
  - High

Emergent Metaphor of Design

- Focus:
  - Internal
  - Team needs

- Processes:
  - Plan-Do-Check-Act
  - Reflectively

- Risk:
  - Low

Prerequisite for holistic planning is a collaborative design process that embraces change. Changes, as highlighted in Agile Software Development: Principles, Patterns, and Practices [1], represent the shift towards agile development approaches that prioritize flexibility and rapid prototyping. Agile methodologies, as advocated in the Agile Manifesto [2], emphasize collaboration, adaptability, and customer satisfaction over strict planning and adherence to rigid processes. By embracing an iterative approach, teams can respond to change and innovate more effectively, aligning with the principles of self-organization and collective intelligence. Leveraging these principles, organizations can foster environments that are conducive to innovation and continuous improvement, thereby enhancing their ability to adapt to the rapidly evolving demands of the digital age.
The metaphor of the holographic organization draws its inspiration from the fact that every fragment or piece of a broken holographic film contains the information required to completely reconstitute the image represented in the whole film [8, 9]. That is, what is observed in each part provides a reflection of the whole. Evidence suggests that the brain’s memory, intelligence, and functionality are distributed across its parts such that damage to some of the parts does not result in complete loss of the overall functionality [9]. This redundancy of functionality, which is not a response to change and question assumptions, as reflected in, for example, the “speculate-collaborate-learn” cycle of Highsmith’s Adaptive Software Development, stressing the importance of flexible/adaptive planning, extensive collaboration, and learning in achieving agility [3]. Practices include continuous code integration, refactoring to improve design and code, reflection workshops and stand-up meetings to detect what worked and what didn’t, and continual feedback from participating stakeholders. These practices facilitate double-loop learning, supporting the principle of “learning to learn” [8]. The agile approach to software development appears to align itself with the principles of holographic organization theory (see Figure 2).

CONCLUSION

The agile methods depart from the traditional orthodoxy of software development. This shift in philosophy is not unusual, as similar patterns of intellectual evolution have emerged in other disciplines. A look at architecture and strategic management reveals that the progression of ideas in them is remarkably similar to conceptual patterns shifts in software design. Even a quick inquiry into the future of software development practices by examining the evolution of design ideas in architecture, strategic management, and other disciplines are even more relevant as system domains extend beyond simpler needs (such as technical functionality) to the complex social aspects of software applications (such as aesthetics, values, human judgment, morals, and ethics).

REFERENCES

According to the Agile Methodology, a software solution or development project can never truly be “complete.” In theory, the requirements, the objectives, and the solutions are always evolving as inputs and resources change. Scrum and Extreme Programming. Now, several types of software development that fall under the broader umbrella of “Agile.” Consistent reflection on how to become more effective when coupled with consistent adjustment based on that reflection leads to the best software. Progress and success will be measured by the caliber of software delivered more than anything else. Realistic development goals will be created to ensure consistency in delivery.