ing. In software engineering, the process models have evolved into several forms that range from highly structured preplanning (waterfalls, spirals, Vs, and CMM) to relatively unstructured agile (XP, SCRUM, Crystal, and evolutionary). No one process is best for every problem.

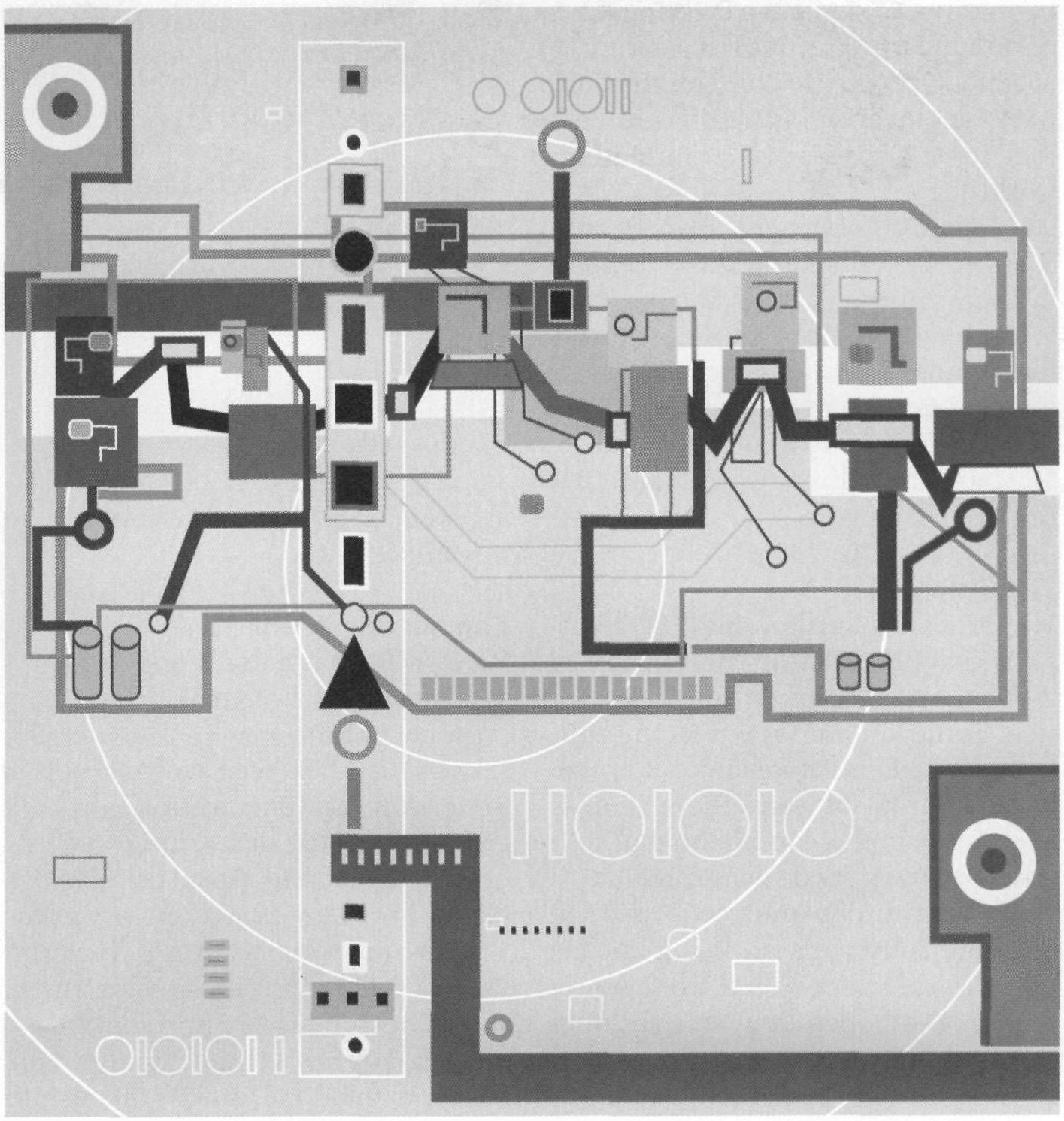
Despite long experience with these processes, none consistently delivers reliable, dependable, and affordable software systems. Approximately one-third of software projects fail to deliver anything, and another third deliver something workable but not satisfactory. Often, even successful projects took longer than expected and had significant cost overruns. Large systems, which rely on careful preplanning, are routinely obsolescent by the time of delivery years after the design started.[2] Faithful following of a process, by itself, is not enough to achieve the results sought by engineering.

Engineering Practice

Gerald Weinberg once wrote, "If software engineering truly is engineering, then it ought to be able to learn from the evolution of other engineering disciplines." Robert Glass and his colleagues provocatively evaluated how often software engineering literature does this.[4] They concluded that the literature relies heavily on software anecdotes and draws very lightly from other engineering fields. Walter Tichy found that fewer than 50% of the published software engineering papers tested their hypotheses, compared to 90% in most other fields.[8]

So software engineering may suffer from our habit of paying too little attention to how other engineers do engineering. In a recent extensive study of practices engineers expect explicitly or tacitly, Riehle found six we do not do well.[5]

▸ *Predictable outcomes (principle of least surprise).* Engineers believe that unexpected behaviors can be not only costly, but dangerous; consequently, they work hard to build systems whose behavior they can predict. In software engineering, we try to eliminate surprises by deriving rigorous specifications from well-researched requirements, then using tools from program verification and process management to assure that the specifications are



met. The ACM Risks Forum documents a seemingly unending series of surprises from systems on which such attention has been lavished. Writing in ACM SIGSOFT in 2005, Riehle suggested a cultural side of this: where researchers and artists have a high tolerance, if not love, for surprises, engineers do everything in their power to eliminate surprises.[6] Many of our software developers have been raised in a research tradition, not an engineering tradition.

▸ *Design metrics, including design to tolerances.* Every branch of modern engineering involves design metrics including allowable stresses, tolerances, performance ranges, structural complexity, and failure probabilities for various conditions. Engineers use these metrics in calculations of risk and in sensitivity analyses. Software engineers do not consistently work with such measures. They tend to use simple retrospective measures such as lines of code or benchmark performance ranges. The challenge is to incorporate more of these traditional engineering design metrics into the software development process. Sangwan gives a successful example.[7]

▸ *Failure tolerance.* Henry Petroski writes, "An idea that unifies all engineering is the concept of failure. Virtually every calculation an engineer performs...is a failure calculation... to provide the limits than cannot be exceeded." There is probably no more important task in engineering than that of risk management. Software engineers could more thoroughly examine and test their engineering solutions for their failure modes, and calculating the risks of all failures identified.

▸ *Separation of design from implementation.* For physical world projects, engineers and architects represent a design with blueprints and hand off implementation to construction specialists. In current practice, software engineers do both, design and build (write the programs). Would separation be a better way?

▸ *Reconciliation of conflicting forces and constraints.* Today's engineers face many trade-offs between conflict-