EDITOR **PHIL LAPLANTE**
The Pennsylvania State University;
plaplante@psu.edu

Lessons From the Father of Software Engineering

Ricardo Valerdi, University of Arizona

The recent passing of Barry Boehm motivated the opportunity to synthesize lessons from his contributions in software engineering. This article summarizes six lessons that impacted how software is designed, built, and managed.

Former Brooklyn Dodgers baseball player Jackie Robinson once said, "A life is not important except in the impact it has on other lives." Barry Boehm, known as the "father of software engineering," was a perfect example of this. His contributions literally changed how software is built—from Beijing to London and everywhere in between. More importantly, he influenced academics and practitioners in the field to think about how to embed more scientific rigor and discipline into software engineering.

Digital Object Identifier 10.1109/MC.2022.3219527
Date of current version: 9 January 2023

It would be impossible to summarize all of Barry's contributions in one article. Instead, it would be more beneficial to identify some of the lessons learned that every software developer and software manager should know. In fact, I strongly believe that Barry's contributions apply beyond the context of software development. Their relevance to product development and technology management is driven by the broad applicability of his work. Here are six lessons I learned from having Barry as my mentor during my doctoral studies and subsequently as a collaborator for the past 20 years. (See also "Barry on the Tennis Court.")

LESSON 1: STRIVE FOR WIN-WIN

Barry's objective function was to make everyone a winner. He referred to this as win-win, or *Theory W*. The idea was that software project managers will be fully successful if and only if they make winners of all of the other participants in the software process: superiors, subordinates, customers, users, maintainers, and so on.[1] If any success critical stakeholder is left out, then the project is at risk of

Barry understood that people are motivated by good will and self-interest.

failing. This expanded focus captured the many views of how we define success for software developments. Rather than emphasizing just performance, Barry valued happy developers, satisfied superiors, and pleased users equally.

The characterization of a manager as a negotiator was innovative at the time because it shifted the emphasis from an organizer of tasks to a "packager of solutions." It also placed a focus on the developer and the user as equal participants. Barry understood that people are motivated by good will and self-interest. Accordingly, it was important to let their value preferences surface to understand what features were most critical. This idea applies to any project that has multiple collaborators. Whether the objective is to build a sports arena or to write a new corporate policy, Theory W reminds us that the most important thing is to ensure which success critical stakeholders need to be satisfied.

LESSON 2: RESOLVE MODEL CLASHES

Sometimes it is impossible to make everyone happy. Barry developed a simple way to think about such "sticky"

BARRY ON THE TENNIS COURT

Barry Boehm was known to many as a humble intellectual—a gentle giant in our field. But if you ever faced him on the tennis court, you would witness an entirely different side of him. With a racquet in his hand, the man was a beast. His style of play was aggressive, and his stamina on the court was on par with a Wimbledon champion because he ran three miles every morning.

When I first met Barry in the early 2000s as a new graduate student in his lab at USC, he was wearing a Wimbledon sweater. I didn't think much of it. I thought he was simply a tennis fan. Barry walked slowly, spoke quietly, and took *veeeeeery* long pauses between thoughts. I asked him if he played tennis, and he confidently said, "I do!" and invited me to play. I thought we would meet at the courts on campus or maybe at a park in Santa Monica. Instead, he invited me to his house because he had his own tennis court in his backyard. That's when I realized my professor was serious about tennis.

Going to someone's house to play tennis is not an everyday occurrence, especially in urban Los Angeles. Nevertheless, I was confident in my tennis game because I had played competitive tennis since I was young. When I showed up at his house, Barry was dressed in old, beat-up tennis attire from the 1980s. His wardrobe featured a partially ripped shirt, shoes with holes in them, and glasses that fit crookedly on his face. I soon learned that this was simply a facade. He was about to rain down aces on me and approach the net on every single point. Every point. Who does that? Barry did. Because Barry learned to play tennis in the 1950s when the racquets were wooden and the only way to hit a forehand was with a long backswing and a long follow-through. One-handed backhands were the norm, and (as Barry demonstrated) the sooner you came up to the net to hit a volley, the more likely you were to win the point.

Barry's game wasn't his only secret weapon. He had home court advantage. This came from knowing where all of the cracks were on the tennis court. You see, Barry's house was near the Santa Monica fault line, which meant that the ground moved over time. This led to sporadic cracks on one side of the tennis court. They were small cracks, but if a tennis ball bounced on one, there was no way of predicting where it would go. Barry seemed to always aim for the cracks and would occasionally hit them. When he did, it was impossible to return the ball. It was an unfair match. An opponent in great physical condition with an aggressive approach and the home court advantage was a difficult one to beat.

During the five years of being Barry's grad student, I did not beat him once. As soon as I defended my doctoral dissertation, I decided it was time for his streak to end. (By this time I had learned where all the cracks were.) But, more importantly, I had learned to never hit it to his forehand because it was his aggressive side. I loved playing tennis with Barry because he loved to compete. Initially, I was embarrassed to lose to someone 40 years older than me, but he was the best player of his generation I had ever played against. When I am his age, I hope to be half as talented, half as humble, and half as competitive. I will dearly miss my professor and tennis partner, but I recognize that the ball is in my court to carry on his legacy.

situations through the *model-based systems architecting and software engineering approach*.[2] Disagreements between parties were reframed into "clashes" between them and represented as different types of models: product models, process models, property models, and success models. Through this reframing of preferences, it became possible for success-critical stakeholders to visualize or reason about the prospective system and its likely effects to better deal with them. These models captured product attributes; the process for developing the system; properties such as cost, performance, or dependability; what it means for the system to be successful; and the means to resolve the many conflicts that occur as the product is architected, designed, developed, tested, and fielded.

An example of a model clash is when a product model aims for high reliability and high performance, while a property model aims for low cost and a short development schedule. Barry's approach gave stakeholders a way to surface their preferences, perform tradeoffs, and come to the negotiating table with specific needs. This facilitated negotiations about things that mattered the most by requiring collaborators to prioritize and reach consensus on what was important. The simplicity of this approach facilitated its use throughout the software community. But model clashes occur in areas outside of software too. As a result, the approach was put to work across systems, hardware, and software communities to resolve conflicts and reach decisions as to what was best for all of those concerned.

LESSON 3: ITERATE TO GAIN KNOWLEDGE AND UNDERSTAND STAKEHOLDER PREFERENCES

Another of Barry's fundamental contributions was the idea that software development should be an iterative process, rather than a specification-driven process. This approach, appropriately called the *spiral model*, was created to overcome the limitations of the waterfall model, which recommended that software be developed in successive stages.[3] Such top-down structured approaches had other difficulties, such as assuming uniform progression of the system's evolution and the inability to accommodate software reuse, among other things.

The innovation of the spiral model is that it reframed software development as an incremental process with multiple milestones along the way. Ultimately, the goal was to identify risks early in the development process and resolve them as early as possible. The spiral model eventually became the universal software development process. Barry once said that he knew he had made it when he read about the spiral model in a Dilbert cartoon. Such an iterative approach can be useful in situations where the end result is not well defined and would benefit from incremental steps that provide information and feedback along the way. Each iteration, or spiral, allows stakeholders to identify risks, experiment with solutions, identify corrective actions, and reevaluate the results, which would ultimately lead to better products.

LESSON 4: A SOLUTION THAT IS TOO EXPENSIVE IS NOT A SOLUTION

Economic analysis techniques were relatively new to software engineering in the 1980s. Limited computing power and memory increased the importance of cost–benefit analyses of software product features. Barry knew that "it is often worth paying for information because it helps us make better decisions."[4] To make this possible, he developed the *constructive cost model* (*COCOMO*). Besides helping people understand the cost consequences of their decisions, the model allowed its users to conduct tradeoffs and come up with a plan that helped them realize the goals set by collaborators. Barry elevated the importance of cost as a critical decision criterion for projects and democratized the ability for people to generate their own cost estimates based on a set of parameters that were known to influence the cost of software.

If two designs have the same functionality but differ in costs, a model like COCOMO can help break the tie.

COCOMO became the standard measuring stick for industry and government to better understand how to keep software development affordable. It also allowed decision makers to quantify the effects of different parameters on cost, such as software reliability, programmer capability, and project schedule. The broad adoption of COCOMO gave rise to the development of algorithmic models that could be used to forecast the cost and schedule of projects in their early stages. This capability is also helpful when comparing possible design alternatives and conducting a wide range of tradeoffs. If two designs have the same functionality but differ in costs, a model like COCOMO can help break the tie. Alternatively, if a project exceeds a budget, then either its forecasted cost or schedule should be adjusted or it should be declared infeasible.

LESSON 5: BE PROBLEM DRIVEN INSTEAD OF METHOD DRIVEN

As a researcher, Barry was not limited to a fixed set of tools or methods. Instead, he was motivated by real-world problems that required robust solutions and an open mindset. The best

example is the COCOMO suite of models, which was motivated by questions that the basic COCOMO model could not answer. The COCOMO model answered two questions: How many software developers do I need to build this system, and how long will they take to complete the project? As the sophistication of software products evolved over time, new development paradigms emerged, the effects of which could not be covered by COCOMO. One of these paradigms was the increased use of commercial off-the-shelf (COTS) software to accelerate development schedules and reduce labor costs. Another was the deployment of agile methods. To address these new challenges, Barry developed new calibrations or other models, like COCOTS.[5]

This important lesson from Barry's work also highlights his versatility as a researcher. By being problem driven, Barry focused on identifying a problem to be solved and then worked diligently to solve it. This resulted in an impactful body of work that has left a lasting legacy in software engineering. Such an approach can apply anywhere there are challenging problems to solve. Being method driven might still be helpful in some situations, but one could become trapped by "having a hammer and only being interested in nails." Barry has demonstrated that being problem driven can be much more gratifying in the long run.

LESSON 6: EMBRACE UNCERTAINTY

Barry's approach to being risk driven in a world with model clashes was centered on the idea that life is full of uncertainties. To illustrate this notion, he popularized the *cone of uncertainty*, which is the idea that uncertainty decreases at an increasing rate as a project progresses.[4] Cost estimates are subject to a high degree of uncertainty at the beginning of a project when information is limited. As more information is obtained from prototypes, user feedback, and iterative development, the uncertainty tends to decrease.

The applications of the cone of uncertainty extend to scenarios where uncertainty must be accounted for in a forecast. This notion is used in hurricane forecasting, where the projected path of a hurricane is bounded by an error cone. Instead of uncertainty decreasing as in software development, uncertainty increases to reflect the error associated with the possible paths of a storm. In both cases, uncertainty is incorporated into a forecast, which leads us to embrace uncertainty by modeling it rather than being intimidated by it.

The underlying premise of the cone of uncertainty is that risk needs to be quantified and tradeoffs must be made to perform this quantification. Such tradeoffs allow "value" to be determined as different parameters are evaluated and tradeoffs are made (cost versus schedule, cost versus reliability, schedule versus personnel availability, and so on). Such tradeoffs are important in light of current development frameworks because they allow decision makers to look at the big picture rather than just at the economics of the situation.

The underlying premise of the cone of uncertainty is that risk needs to be quantified and tradeoffs must be made to perform this quantification.

Just as Babe Ruth was known as the greatest baseball player of all time, Barry is considered to be the best software engineer of all time. Barry was a Fellow of multiple professional societies, including IEEE, and he had the power to convene thought leaders from around the globe to move the field forward. Barry was an intellectual giant: every one of his contributions was a home run. Most important of all, Barry was the Most Valuable Person. C

REFERENCES

1. B. W. Boehm and R. Ross, "Theory-W software project management principles and examples," *IEEE Trans. Softw. Eng.*, vol. 15, no. 7, pp. 902–916, Jul. 1989, doi: 10.1109/32.29489.
2. B. W. Boehm, "Escaping the software tar pit: Model clashes and how to avoid them," *SIGSOFT Softw. Eng. Notes*, vol. 24, no. 1, pp. 36–48, Jan. 1999, doi: 10.1145/308769.308775.
3. B. W. Boehm, "A spiral model of software development and enhancement," *ACM SIGSOFT Softw. Eng. Notes*, vol. 11, no. 4, pp. 14–24, Aug. 1986, doi: 10.1145/12944.12948.
4. B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1981.
5. B. W. Boehm et al., *Software Cost Estimation With COCOMO II*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2000.

RICARDO VALERDI is a professor and head of the Department of Systems & Industrial Engineering at the University of Arizona, Tucson, AZ 85721 USA. He completed his Ph.D. at the University of Southern California under Barry Boehm's mentorship and played many tennis matches against him. Contact him at rvalerdi@arizona.edu.