# The Importance of Teams

by Grady Booch

As Gerald Weinberg pointed out over 25 years ago in his seminal work, The Psychology of Computer Programming, "computer programming is a human activity."1 When you are up at 2 AM trying to stamp out an elusive bug, programming does indeed look like an isolated human activity. However, there's a big difference between cutting code and shipping products: deploying quality software is a team sport that requires a group of people with a variety of skills working together toward a common goal.

Associated with his work on the Capability Maturity Model (CMM) for development organizations, Watts Humphrey observed that "the history of software development is one of increasing scale. Initially, a few individuals could hand craft small programs; the work soon grew beyond them. Teams of one or two dozen professionals were then used, but success was mixed. While many organizations have solved these small-system problems, the scale of our work continues to grow. Today, large projects require the coordinated work of many teams. Complexity is outpacing our ability to solve problems intuitively as they appear."2

There are perhaps two major forces that drive this complexity. First, there is technology push, wherein the ever-declining cost of hardware and the growing availability of high-speed networks makes it possible to develop automated solutions that, even a year ago, would not have been economically feasible. Second, there is the societal pull, wherein individuals and organizations have come to rely upon such automation, and in so doing have developed an insatiable demand for systems that are better, faster, and cheaper. Combine that with a world-wide shortage of skilled software developers, and the net result is that the typical development team is being asked to do more with less: more features and more quality with less resources and less time.

Weinberg goes on to note that "for the best programming at the least cost, give the best possible programmers you can find sufficient time so you need the smallest number of them."3 That is certainly sound advice: all things being equal, it's better to have a small team than a large one, and to be relatively unconstrained by schedule. But, as Fred Brooks points out, "the dilemma is a cruel one. For efficiency and conceptual integrity, one prefers a few good minds in doing design and construction. Yet for large systems one wants to find a way to bring considerable manpower to bear, so that the product can make a timely appearance."4

Most problems are sufficiently complex that they simply require a lot of hard work and sustained labor, more than can be carried out by a single developer working in isolation.5 However, you cannot just expect a project to succeed by staffing it with superstars and arming them with powerful tools. Walker Royce points out that, although hiring good people is important, it's far more important to build a good team.6 He goes on to explain that balance and coverage are essential characteristics of such a team, and describes this by analogy. "A football team has a need for diverse skills, very much like a software development team. There has rarely been a great football team that didn't have great coverage: offense, defense, and special teams, coaching and personnel, first stringers and reserve players, passing and running. Great teams need coverage across key positions with strong individual players. But a team loaded with superstars, all striving to set individual records and competing to be the team leader, can be embarrassed by a balanced team of solid players with a few leaders focused on the team result of winning the game."7

In the context of software development, "winning the game" means developing and deploying quality software in a predictable and sustainable fashion. As Barry Boehm demonstrates in COCOMO (a model for software cost estimation), the capability of the team has the greatest impact upon productivity.8 Tom DeMarco and Tim Lister go on to note that, within a team, an organization's most productive people will tend to outperform its least productive by a factor of 10:1.9

Historically, most advances in software development languages and tools have focused on improving the productivity of the individual developer. This is not to say that such advances are unimportant; I'd rather have a fast compiler than a slow one. However, given the importance of teams to modern software development, such advances in individual productivity have diminishing returns relative to winning the game. As such, it makes sense to turn our attention to the software development team, and ways to prove its productivity.

## Organizing the Software Development Team

Drawing upon his experience inside Microsoft, Steve McConnell notes that "it takes more than just a group of people who happen to work together to constitute a team. In their book The Wisdom of Teams, Katzenbach and Smith define a team as 'a small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable.'"10 He goes on to enumerate the characteristics of a hyperproductive team:11

•a shared, elevating vision or goal•a sense of team identity•a results-driven structure•competent team members•a commitment to the team•mutual trust•interdependence among team members•effective communication•a sense of autonomy•a sense of empowerment•small team size•a high level of enjoyment

There are many different ways to organize such a team: the business team, the chief-programmer team, the skunkworks team, the feature team, the search-and-rescue team are among those that McConnell identifies.12 Although no one organization is optimal for every team or problem domain, the surgical team (as Brooks calls it13), also known as the chief programmer team, is perhaps the most common and most effective.

In this organization, the central player in the development organization is the architect, who is responsible for the "conceptual integrity of all aspects of the product perceivable by the user."14 Don't confuse the role of the architect and the project manager, however, as they encompass very different activities.15 A team's architect is responsible for designing the system, whereas the project manager is responsible for designing the team and making it possible for them to do their job.

Within a team, you should choose an architect who possesses a vision large enough to inspire the project to great things, the wisdom born from experience that knows what to worry about and what to ignore, the pragmatism necessary to make hard engineering decisions, and the tenacity to see this vision through to closure.16 On small projects, one architect is sufficient. On larger projects, architects are a recursive feature; different subsystems will require their own architect, with the overall conceptual integrity of the system being maintained by a single architect or, more commonly, a small team of architects.

Surrounding these architects at each level in the system are application developers, people who love to code and who are able to turn abstractions into reality.

Thus, the center of gravity of the development team should be formed by a tight grouping of the organization's project manager, its architect, and its application developers. Jim Coplien, in his study of hyperproductive organizations, has found that many such teams exhibit this common pattern or organization.17

These three skill groups are necessary but, as it turns out, insufficient. The successful deployment of any complex system requires a variety of other skills: toolsmith, quality assurance, system integration, build and release, and analyst are some of the identifiable roles you'll need to fill on most projects. Capers Jones notes that "the software industry reached the point of needing specialists as long ago as twenty years.... For some large enterprises, there may be more than one hundred different occupational groups within an overall software organization."18 In large projects, you'll have specific individuals for each such role; in smaller teams, each individual will end up playing multiple, simultaneous roles.

## Improving Team Productivity

Staffing a project with the right people who have the right skills is important, but that alone does not explain the differences in productivity one sees among such teams. In this context, DeMarco and Lister speak of a "jelled team" which they define as "a group of people so strongly knit that the whole is greater than the sum of the parts. The productivity of such a team is greater than that of the same people working in unjelled form.... Once a team begins to jell, the probability of success goes up dramatically."19

Essential to the formation of jelled teams is this precondition: a project must honor and respect the role of every one of its developers. This means that each project must recognize that its developers are not interchangeable parts, and that each brings to the table unique skills and idiosyncrasies that must be matched to the needs at hand and calibrated within the organization's development culture. This is one of the five basic principles of software staffing that Boehm describes: "fit the tasks to the skills and motivation of the people available."20

DeMarco and Lister suggest a simple formula for creating a jelled team:21

•get the right people•make them happy so that they don't want to leave•turn them loose

They go on to note ways to develop an organizational culture that encourages jelled teams to develop and flourish:22

•make a cult of quality•provide lots of satisfying closure•build a sense of eliteness•allow and encourage heterogeneity•preserve and project successful teams•provide strategic but not tactical direction

Teams and Tools

When projects were simple and teams were small (often involving teams of one), an organization could get by with only the simplest of tools: an editor, a compiler, and a linker would be quite sufficient for many problems. Add a debugger, and you were really rocking.

Amazingly, lots of organizations still get by with only this minimalist set of tools. However, in the context of contemporary software development, that's akin to banging rocks together to light a fire: you can do it, but it's not particularly efficient. Rather, as the complexity of your project grows, you must consider other, more targeted tools, such as tools for requirements capture, visual modeling, configuration management and version control, performance analysis, and testing.

Personal integrated development environments are important in making the individual programmer productive. Indeed, there's been a long history of maturation of such tools, and their advances have been essential in enabling the creation of larger and more complex systems. However, there are limits to the cool features you can provide to the individual developer, and it would appear that we as an industry are getting close to those limits. Indeed, there is only so much you can do to help an individual developer bang out code faster.23

A recent trend in the industry has been the integration of such tools, tools that individually address point problems but that collectively cover the full spectrum of lifecycle activities. That's certainly a predictable advance, but it's not necessarily the most important one. A greater improvement in the overall productivity will only come from tools that empower the team as a whole. You must integrate your team, not just your tools.

Ed Yourdon points out that "the only way such a tool could be a silver bullet is if it allows or forces the developers to change their processes."24 In other words, while it's important to supply your development team with good tools, you'll only see a state change in your team's productivity if you apply tools that encourage and enforce a sound development process.

The Next Generation of Team Tools

Yourdon's comments suggest three trends that will likely drive future software development tools.

First, there is the need to unify a team's tools around a common process and a common set of artifacts. Development teams create and modify artifacts such as requirements, plans, models, code, components, tests, and so on. They employ tools to create and modify those artifacts. Insofar as those tools are clumsy to use or get in the way of manipulating those artifacts, they detract from the primary focus of the development team, namely, deploying quality systems in a predictable and sustainable fashion. That means that, across the set of tools used by the development team, they must share and preserve a uniform understanding of the semantics of these artifacts and the activities that manipulate them.

Second, there is the need to optimize a team's tools to the different skill sets that exist within the team, since many of a project's artifacts will be created and manipulated by different stakeholders. For example, a requirement may be created by an end user, elaborated upon by the project's architect, and referenced by the project's quality assurance personnel. Each one of these stakeholders has a different view into the artifacts of the project. As such, it would be suboptimal to provide a "one size fits all" development environment. Rather, it's far better to provide a tool set that permits different, simultaneous views into a project's artifacts, optimized to the needs of each individual stakeholder.

Third, there is the need to simplify the delivery of these tools, especially for development teams that are distributed in time or in space. Practically, this means we'll likely see such toolsets become Web-centric, since the Web is the quintessential common vehicle for providing access to and visualization of information. Already, we find projects that use the Web as a repository for all of their project's artifacts. Open communication is a key enabler in forming a jelled team, and making all such artifacts visible to the project facilitates that kind of communication.

## Growing a Team

All that being said, how do you grow a team? Even if you've loaded up your team with all the latest and greatest tools and techniques, what must you do next to turn a disjoint set of individuals into a jelled team whose productivity is greater than the sum of its parts?

There are three pragmatic techniques that I've seen work.

First, identify clear roles and responsibilities suitable to your development culture and necessary for your particular domain, and then match individuals with the right skills to those roles.25 For example, perhaps the most important role you need to identify is that of architect: an architect is the person or persons responsible for establishing the significant design decisions for the system and for creating and validating a suitable architectural style. An architect may not be your fastest or most clever programmer, but he or she must certainly be your most wise.

Second, consider the essential artifacts of your project, and organize your team around the set of activities that produce and manipulate those artifacts. Clearly, the most important artifact of the software project is the running system itself, but that alone is insufficient: the team must create a scaffold of other artifacts around that system in order to build it in an efficient and predictable fashion. This means selecting what other work products you need — such as software architecture documents, test plans, releases, and so on — and then establishing a plan for growing and evaluating those artifacts. For example, in an iterative style of development, it's important to drive each iteration according to essential use cases (which specify the desired behavior of the system and additionally serve as test cases) and according to project risk (which changes with each iteration, and may be manifested as technical, economic, or business risk). By focusing on artifacts such as these in a controlled and measured way, you create an environment that encourages your team to drive their work to closure and to focus on the most important things they can do at each moment to mitigate the risks of the project.

Third, leverage tools that let each individual manipulate these artifacts in a manner appropriate to their specific role and consistent with other roles, and in a manner that reduces the interference between individuals. This is what the emerging generation of team tools is all about: providing an environment that encourages individual skills and enables those individuals to work productively and cooperatively.

## Wrapping Up

These are indeed interesting times. The challenges of software development are certainly not going to go away, for we as an industry are continually being driven to do more with less. Methods and processes help; so do languages, frameworks, and point tools. However, software development is ultimately a human endeavor, and as such it's ultimately the efforts of the software development team that enable us to deliver quality systems in a predictable and sustainable fashion. Tools that unify, optimize, and simplify the work of that team represent the next state change in helping create jelled teams.

## Bibliography

Boehm, B. 1981. Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall.

Booch, G. 1996. Object Solutions: Managing the Object-Oriented Project. Menlo Park, CA: Addison-Wesley.

Brooks, F. 1995. The Mythical Man-Month: Essays on Software Engineering, anniversary edition. Reading, MA: Addison-Wesley.

DeMarco, T. and Lister, T. 1987. Peopleware: Productive Projects and Teams. New York, NY: Dorset House.

Gilb, T. 1988. Principles of Software Engineering Management. Wokingham, England: Addison-Wesley.

Glass, R. 1998. Software Runaways: Lessons Learned from Massive Software Project Failures. Upper Saddle River, NJ: Prentice Hall.

Humphrey, W. 1989. Managing the Software Process. Reading, MA: Addison-Wesley.

Jones, C. 1996. Patterns of Software Systems Failure and Success. London, England: Thomson Computer Press.

McCarthy, J. 1995. Dynamics of Software Development. Redmond, WA: Microsoft Press.

McConnell, S. 1996. Rapid Development: Taming Wild Software Schedules. Redmond, WA: Microsoft Press.

Royce, W. 1998. Software Project Management. Reading, MA: Addison-Wesley.

Shneiderman, B. 1980. Software Psychology: Human Factors in Computer and Information Systems. Cambridge, MA: Winthrop Publishers.

Weinberg, G. 1971. The Psychology of Computer Programming. New York, NY: Van Nostrand Reinhold.

Yourdon, E. 1997. Death March. Upper Saddle River, NJ: Prentice Hall.

References

1. Weinberg, p. 3

2. Humphrey, p. vii.

3. Weinberg, p. 69

4. Brooks, p. 31

5. Booch, p. 190

6. Royce, p. 43

7. Royce, p. 43

8. Boehm, p. 642

9. DeMarco, p. 45

10. McConnell, p. 275

11. McConnell, pp.278-279

12. McConnell, pp. 304-313

13. Brooks, p. 29

14. Brooks, p. 256

15. Booch, p. 200

16. Booch, p. 197

17. Booch, p. 207

18. Jones, P. 215

19. DeMarco, p. 123

20. Boehm, p. 669

21. DeMarco, p. 93

22. DeMarco, p. 151

23. As an aside, I've often said that the best way to accelerate software development is simply by writing less software. That's why object-oriented techniques and component-based development are useful: the former encourages you to write less code via inheritance and abstractions that form a balanced set of responsibilities, and the latter encourages you to reuse and adapt existing components and frameworks rather than writing your own from scratch.

24. Yourdon, p. 183

25. For example, the Rational Unified Process specifies a common set of roles, which it calls "workers".

Rational's solution to team development ... Rational Suite Rational Software Corporation is addressing the topic of team development through Rational Suite, a complete software lifecycle development solution designed for both Microsoft Windows and UNIX platforms. Through this solution, organizations can easily adopt a single set of best practices for all of their project teams; a philosophy that is crucial for any organization committed to producing the highest quality software possible in today's changing, business-critical software development climate.

One of the best features about Rational Suite is its versatility, as it adds value whether customers use all or some of the suites. Developers already using some products within the Rational Suite will find that the deployment of the entire suite empowers the entire project team, from analyst to tester. Rational Suite also provides an ideal way for developers new to Rational to give their organization an integrated suite of proven, market-leading products that lay the groundwork for automating the entire development lifecycle. Everything from the initial acquisition and deployment process, through to training and support is made easier with the convenience and comfort of working with one world-leading organization.

Rational is the first company to deliver an integrated solution for the entire cross-functional software team. According to Rational VP Eric Shurr, "We are very proud of the philosophy of team development that is at the heart of Rational Suite. Through this complete lifecycle development solution, you can unify analysts, developers, and testers by breaking down the barriers that normally exist between cross-functional teams. You can optimize each practitioner's performance by delivering the right market-leading tools to each member of the team. Plus you can simplify the solution - and lower your total cost of ownership."