A better way to manage application development and maintenance work

Inflexible staffing models are straining application developers' ability to keep up with rising demand.

Daniel Alsén, Rajeev Jain, and Krish Krishnakanthan

Most organizations deploy multiple teams to develop and maintain software applications, a function that's known as application development and maintenance (ADM). Often these teams are organized around projects or application expertise. But demand for application work usually doesn't sync with such fixed team structures. The unbalanced flow means that some project groups may find themselves less busy—while others can barely keep up with the influx. The challenges many face are likely to rise: demand for ADM services is expected to grow at an annual rate of 4 to 5 percent through 2013 as more products, processes, and functions become automated or application driven.

The ADM department of one large high-tech company, for instance, found itself in perennial catch-up mode. Business in the services division was booming, but release rates for applications had fallen off steeply. With deliverables pending at major accounts, developers faced constant pressure to boost productivity. To quicken the pace, managers parceled out jobs on a first-come, first-served basis. The department's credo was that if you had the skill, you picked the job off the pile and got to work. In practice, quick-turn projects often got stuck in the pipeline while developers—sometimes with only basic skills in the required programming language—worked through the kinks of bigger, more complex tasks. Burnout and rising attrition exacerbated staffing constraints.

As ADM organizations such as this one muscle their way through steadily expanding workloads, the cracks are not only becoming evident but are also

Takeaways

By splitting application work by client or account, traditional ADM staffing models create silos that aggravate capacity constraints.

For better results, break projects down by complexity, duration, and skill.

Assign simple, quick-turn elements to a large, integrated staffing pool and route complex tasks to a smaller group of specialists to better align resources with the demand curve.

Aggregate programmers who have the same or related programming-language skills into larger clusters that can operate across internal company borders for additional efficiencies.

affecting broader organizational performance. Misaligned workflows mean that staff must often sprint back and forth between projects to meet escalating priorities. That scrambling creates spot shortages in capacity and makes it hard to estimate project time lines. The result: cost and deadline overruns are increasingly the norm.

One way to ease these capacity constraints is to restructure ADM work fundamentally. That requires breaking down projects and sorting them by complexity, skill, and duration, as well as abandoning traditional account or program silos. This article describes how companies can meet that goal.

Hitting the wall

Large ADM service organizations often divvy up application work by client or account, but these traditional staffing models can aggravate capacity constraints. Workload volumes often swing dramatically from one account to another, thus lumping some staff members with a half dozen or more projects while others manage just two or three. Technical-development work (for instance, projects involving Java, C++, Web/HTML, and mainframe programming) is frequently siloed, even though many developers have crossover skills that could ease the pressure on overtaxed teams.

Within project teams, a constant stream of requests can reduce workflow management to primitive firefighting as developers manage assignments spanning a range of time horizons and design complexity. Overlapping deadlines and frequent project interdependencies can make it hard for developers to see assignments through to completion without being interrupted. Experienced programmers and designers who can churn out complex application algorithms and undertake demanding development tasks find themselves consumed instead with a series of less difficult but urgent projects.

These problems can make delivery schedules a guessing game. In the absence of defined workloads, even helpful estimation techniques (such as story, use case, or function points) may be of limited use.[1] Many managers describe the process as "working blind."

A new pathway to managing capacity

Our client work shows that ADM groups can gain greater control and visibility if they break down projects by complexity, duration, and skill and then aggregate programming silos into new, integrated clusters.

Sort work by skill and complexity

Many organizations normally assign beginning-to-end responsibility for each application to a specific customer team. We find, however, that they would often see faster results by giving simple, quick-turn elements to a large, integrated staffing pool and routing complex tasks to a smaller group of specialists. That structure lets the larger group narrow its mission to maximizing throughput (that is, to cranking things off the assembly line faster) and frees the smaller group to concentrate on "long cell" jobs—for instance, ongoing development or maintenance efforts requiring more features, customization, and expertise and spanning a longer time horizon.

Backing this idea is the fact that more than 80 percent of applications in the United States today involve small, low-complexity tasks (defined as having fewer than ten function points) and require relatively basic skills to complete (Exhibit 1). By using staffers accordingly—assigning 70 to

[1] Story points draw on roundtable discussions to estimate an application's size and time to complete. Use case points apply a formula-based approach to do the same thing. Function points categorize user requirements by their relative complexity and by type of function to estimate a project's size and duration.

80 percent of them to low-complexity tasks and the remainder to more difficult ones, for instance—managers can better align resources with the demand curve. That approach also matches the typical skill distribution in most ADM organizations. In our work with clients, expert employees typically make up about one-third of the total staffing pool.

Using this approach, one ADM organization found a way to ease what had become a constant backlog of demand. By allocating low-complexity work to junior staff, the group freed specialists to tackle more complex assignments. Separating work by complexity also helped it fine-tune delivery estimates. The group generally expected to complete tasks with fewer than ten function points—about 80 percent of project volume—in no more than four weeks (and sometimes in less than two). More difficult projects had correspondingly longer timelines.

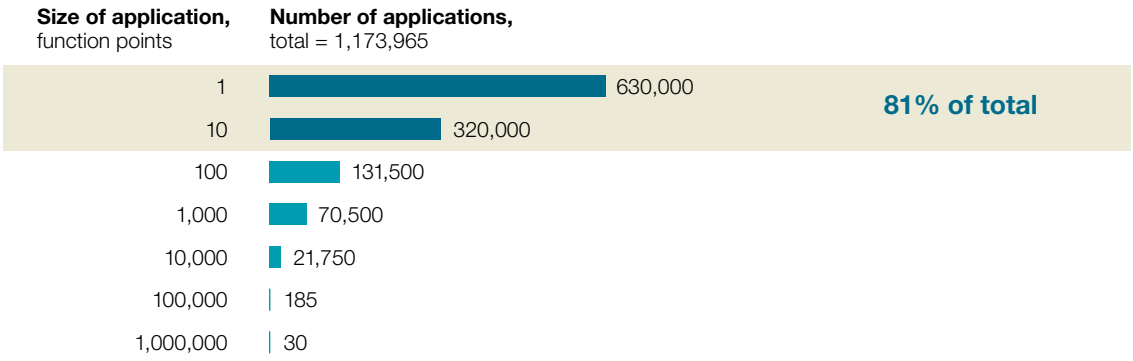
Create integrated programming clusters

ADM groups can gain additional efficiencies by aggregating programmers who have the same or related programming-language or software skills into larger clusters (that is, centers of excellence) that can operate across internal company borders. That kind of bundling can improve scale by making it easier to move resources around as some projects wind down and others begin.

One ADM manager, for example, inventoried the employee skill base and was surprised to find that he could reassign a large cross section of programmers to form such a shared-services team. Rather than having a number of Java and COBOL teams working in parallel for separate business lines, he reasoned, it would make more sense to combine resources, providing for greater sharing of skills and helping to balance tasks. Taking that approach, the manager pooled 11 fragmented programming silos into four integrated system-

Exhibit 1
More than 80 percent of applications in the United States consist of small, low-complexity tasks.

Estimated level of complexity in US application software, 2010



Source: "Using function point metrics for software economic studies," Jan 27, 2010, presentation by Capers Jones

One ADM organization found a way to ease what had become a constant backlog of demand. By allocating low-complexity work to junior staff, the group freed specialists to tackle more complex assignments.

development clusters, leaving only Web/HTML programmers and a catch-all "other" bucket of developers to stand alone. Reducing the number of disparate work streams created greater visibility into the activity of the ADM organization, allowing it to assess the relative complexity and duration of different jobs and to assign resources accordingly (Exhibit 2).

What ADM leaders should do

In our experience, the following four steps can guide executives in realigning their capacity.

- *Identify the complexity and time horizons of incoming demand.* Filter and analyze incoming work to identify recurring and standard tasks across groups and domains. Using this analysis, create a table (complexity catalog) that describes the typical resolution time and complexity level for each task. This table can be used in combination with team "poker planning,"[2] in which joint consensus estimates are created using story points. The end result is a clearer view of the incoming work by complexity.

- *Identify the work and skill sets most commonly in demand.* Conduct discussions with leaders of development teams to determine the key functional, application, technical, and supporting skills required to manage incoming demand. To capture shifts in resource needs, include a review (based on technologies used) of future development road maps.

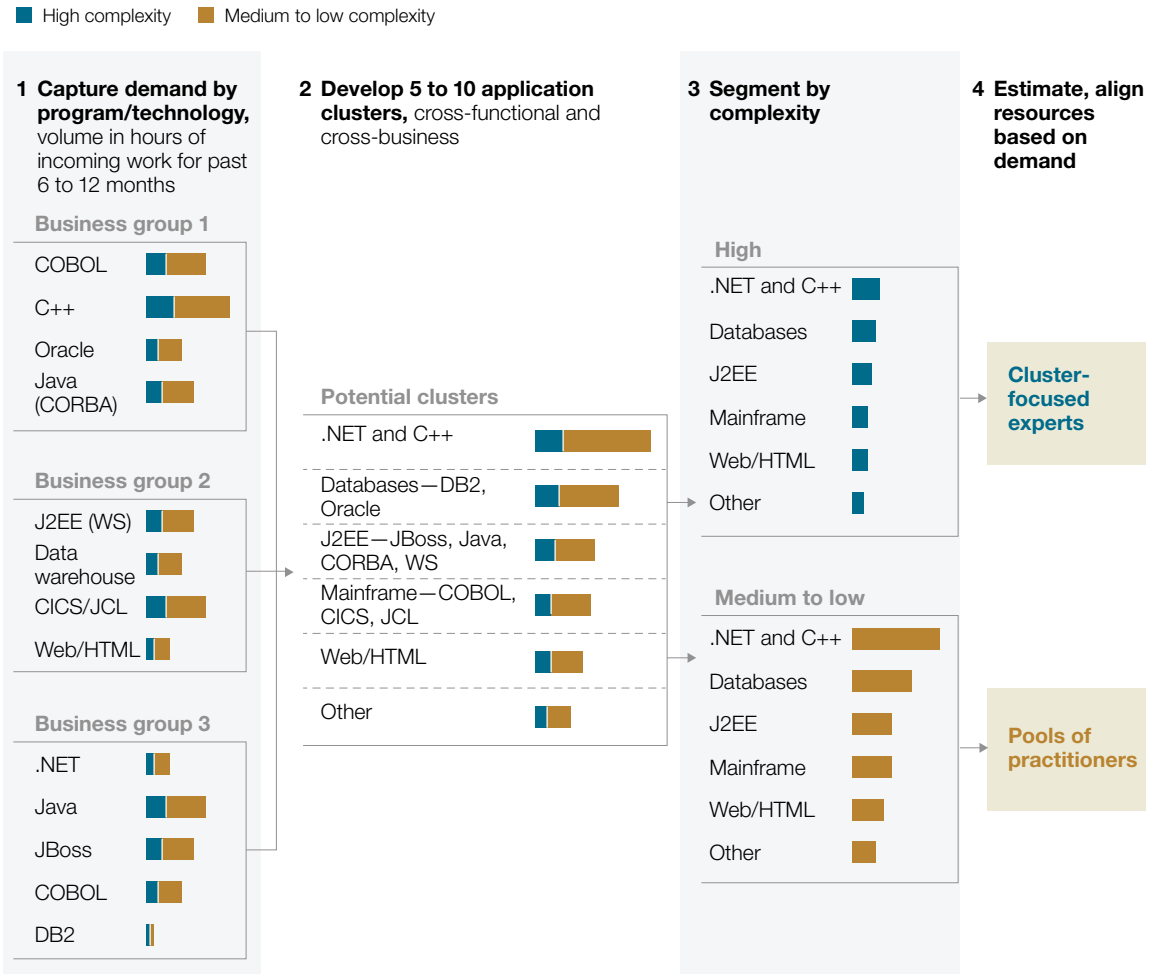
- *Use a skill matrix to chart baseline capacity and identify gaps.* To improve standards throughout your company, conduct an inventory of skills in which employees rate theirs on a clearly defined scale from 0 (no skills) to 4 (sufficient skills). Such an inventory will help you determine where additional training is needed to broaden the skill base and to create a larger pool of staffers equipped to tackle high-demand projects.

- *Pool resources.* Create resource pools that group related programming skills across divisions and groups. Assign tasks by complexity to increase resource utilization and to achieve scale effects. Use the incoming-demand analysis and the skill matrix to balance the workloads of full-time employees.

The high-tech company mentioned at the outset of this article used this approach to get out from the corner it had wedged itself into by taking on more jobs than its ADM structure could manage. It began by dissolving smaller account teams and moving those resources into larger staffing pools.

[2] Poker planning is a consensus-based approach to estimating the complexity of projects.

Exhibit 2
Fragmented programming silos can be transformed into integrated system-development clusters.



Within the pools, it created two work groups: one focused on processing simple tasks (file cleanup and password resets), the other charged with more complex assignments (such as performance tuning and coding changes).

Managers and employees balked at the proposed changes initially. Most were skeptical about the idea that specific applications could be compartmentalized so easily. Junior staff worried that the new system would assign them to work on programs where they would use only the most basic skills. Managers feared that quality issues would skyrocket and that projects would return to square one when they came back for rework.

To get around those concerns, the project leaders dug out ticket data from the previous 12 months

and used this information to show that nearly 70 percent of all incoming work required little or no specific application knowledge. The majority of tickets concerned simple elements, such as system restart and report generation—tasks that required only basic Unix scripting and Structured Query Language (SQL). The remaining 30 percent went to long-cell groups that could handle more complex projects.

To gain even more fine-grained information on the expertise of the staff, managers conducted a skill inventory, using a five-point scale to rank employees on a variety of technical and business capabilities. Staff members with the highest rankings were assigned to the complex-work tier, and the rest to lower ones—a change that generated 23 percent more capacity. To avoid the interruptions caused by cycling work back and forth between groups, the short- and long-cell teams received different deliverables. Before the change, one team might be tasked with developing a new user interface from start to finish. Under the new arrangement, the short-cell team would work through a series of small, individual subcomponents—for instance, changing the color of a screen from yellow to blue. Meanwhile, the long-cell team would focus on more intricate requirements, such as creating a new billing format.

That structure made it easier to estimate delivery schedules. Managers broke overall projects into three waves of work, each scheduled to take four weeks. Because teams worked on discrete elements, quick-turn fixes could be released as they became available rather than getting stuck waiting for a whole project to be delivered "all in one," as before. Customers preferred the new system; knowing that they would receive their blue screen in October and their new billing system in January made it easier for them to plan. Although it took the company four months to reorganize, improvements began after only nine or ten weeks. With capacity better balanced, quality also improved. Maintenance and development errors are now down by 12 to 15 percent.

• • •

ADM organizations are on the hook to process higher volumes of work at a faster clip, so they must move away from individual project teams to aggregated clusters that facilitate better matchmaking between work and worker. That approach can lift the blindfolds from managers, giving them a chance to home in more closely on performance metrics that can improve costs, times to market, and customer satisfaction.

Daniel Alsén (Daniel_Alsen@McKinsey.com) is an associate principal in McKinsey's Stockholm office, **Rajeev Jain** (Rajeev_Jain@McKinsey.com) is a consultant in the Chicago office, and **Krish Krishnakanthan** (Krish_Krishnakanthan@McKinsey.com) is a principal in the New York office.