AUGUST 2013

BUSINESS TECHNOLOGY PRACTICE

Enhancing the efficiency and effectiveness of application development

Software has become critical for most large enterprises. They should adopt a reliable output metric that is integrated with the process for gathering application requirements.

Michael Huskins, James Kaplan, and Krish Krishnakanthan

Most large companies invest heavily in application development, and they do so for a compelling reason: their future might depend on it. Software spending in the United States jumped from 32 percent of total IT corporate investment in 1990 to almost 60 percent in 2011[1] as software gradually became critical for almost every company's performance.[2] Yet in our experience, few organizations have a viable means of measuring the output of their application-development projects. Instead, they rely on input-based metrics, such as the hourly cost of developers, variance to budget, or percent of delivery dates achieved. Although these metrics are useful because they indicate the level of effort that goes into application development, these metrics do not truly answer the question: how much software functionality did a team deliver in a given time period? Or, put another way, how productive was the application-development group?

Flying blind

With big money and possibly the company's competitiveness at stake, why do many application-development organizations fly blind without a metric in place to measure productivity?

First, with every metric comes some level of overhead to calculate and track that metric. With some metrics, the overhead has proved larger than the benefits afforded by them. The

[1] "Private fixed investment in equipment and software by type," table group 5.5.5, *Concepts and Methods of the US National Income and Product Accounts*, US Bureau of Economic Analysis, November 2011.

[2] For more information, see Hugo Sarrazin and Johnson Sikes, "Competing in a digital world: Four lessons from the software industry," *McKinsey on Business Technology*, Number 28, Winter 2012, mckinsey.com.

Takeaways

Few organizations have a way to measure the output of their application-development projects. We believe the best solution is to combine use cases with use-case-point metrics.

Use cases describe the users who interact with the application—and how the application interacts with them. Use-case points, which are based on data captured from these interactions, can be calculated in less than a day, even for large projects, and refined as more requirements are specified.

Organizations that have successfully adopted this approach have started with a pilot involving several teams. What is crucial for the acceptance of these metrics is how leadership uses them; if they're only used to reward or penalize developers, serious resistance is likely.

second reason is that in many application-development organizations there is a lack of standardized practices for calculating metrics. For example, it is difficult to deploy output measurements if application teams are following different approaches to capturing functional and technical requirements for their projects. Finally, and perhaps most important, there is often a certain amount of resistance from application developers themselves. Highly skilled IT professionals do not necessarily enjoy being measured or held accountable to a productivity metric, especially if they feel that the metric does not equitably take into account relevant differences among development projects. As a result, many organizations believe there is no viable productivity metric that can address all of these objections.

Although all output-based metrics have their pros and cons and can be challenging to implement, we believe the best solution to this problem is to combine use cases (UCs)—a method for gathering requirements for application-development projects—with use-case points (UCPs), an output metric that captures the amount of software functionality delivered.

For most organizations, this path would involve a two-step transformation journey—first adopting UCs and then UCPs. While there might be resistance to change from business partners and, not least, application developers, we believe the journey is well worth the effort.

Use cases

In addition to lacking a viable methodology for measuring productivity, organizations often don't have a robust way to gather and organize functional and technical requirements for application-development projects. Instead, they list requirements in what often amounts to little more than a loosely structured laundry list. Organizations may have used this laundry-list approach over a long period of time, and it thus may be deeply entrenched.

As a result, these organizations find it difficult to fully and accurately capture requirements and align them with the needs of their internal or external business clients. Their application-development projects tend to suffer the inefficiencies of shifting priorities, last-minute change requests, and dissatisfied business users. In our experience, these changes often amount to cost overruns of 30 to 100 percent.

We believe use cases provide a logical and structured way to organize the functional requirements of an application-development project. Each use case is a description of a scenario under which the user of an application interacts with that application. For example, a UC for an online-banking application might describe each of the steps that a bank customer follows to log into her account and check the balance of available funds, as well as the transactions involved when that application calls on a database to pull up the stored information.

Another use case for that same application might involve the customer transferring funds from her checking to savings account. More specifically, UCs describe "actors"—the human users or systems that interact with the application in question. UCs also describe "transactions," or how the application interacts with actors and performs a function. Related UCs can be logically organized into sections and chapters with a table of contents so that developers and their business clients can understand the overall structure of the application.

By focusing first on business objectives and the functional requirements of applications rather than on the technical requirements, both business leaders and application developers find UCs easy to understand. Technical requirements and design choices can then be organized around UCs. This structure expedites the requirements-gathering phase of the software-development life cycle. It also lowers the risk of failing to incorporate the functionality required by the business and thereby reduces the amount of costly change requests and rework during the subsequent design and build phases. UCs also make it easier to write functional test cases—and thus expedite the testing process on the back end of development.[3]

Use-case points

Use-case points, as the name implies, are derived from the information captured in use cases. UCP calculations represent a count of the number of transactions performed by an application and the number of actors that interact with the application in question. These raw counts are then adjusted for the technical complexity of the application and the percentage of the code being modified.

The members of one application-development group recently calculated the UCPs for 12 of their completed projects. The leader of that group, who was intimately familiar with the 12 projects, also independently gave each project a relative score representative of the software functionality delivered. The high correlation between the UCP calculations and the leader's scores (greater than 80 percent) suggests that UCPs are highly reliable in measuring output and can be used to accurately and equitably measure productivity across teams (exhibit).

Based on our experience, productivity across application-development teams can differ by more than 50 percent and often by as much as 100 percent. UCPs accurately measure software functionality to within 10 to 15 percent. Consequently, the accuracy of UCPs is more than sufficient to help determine the productivity of teams.

Moreover, UCPs do not take a lot of training to calculate, and the calculations can be completed in less than a day even for large projects. UCPs can be calculated early in a project's life cycle and then refined as more requirements are specified and more of the design work is completed. As a result, they are useful for project planning, in-flight performance management, and retrospective performance evaluation.

In general, UCPs are applicable to waterfall[4] development and can be used by teams following agile[5] methodologies as long as the agile teams use UCs to gather requirements. UCPs, because they are simple to calculate, can also be easily rolled out across an organization. (For a review of alternative methods to measure output, see sidebar, "Pros and cons of four output metrics.")

The transformation challenge

Organizations that have successfully adopted use cases and use-case points have usually started with a pilot that may involve several teams and a portfolio of new projects on which to test the new approach. The organization will need to design the processes and tools to make use cases and use-case points operational. For example, the organization will need to address such questions as what template or tool the team should use for capturing UCs and calculating

[3]The typical software-development life cycle usually encompasses seven phases: conception, analysis (requirements gathering), design, build, testing, release, and maintenance, although different organizations have different names and may group the phases differently. Use cases thus directly benefit several of the most costly and time-consuming phases: analysis (requirements gathering), design, build, and testing.

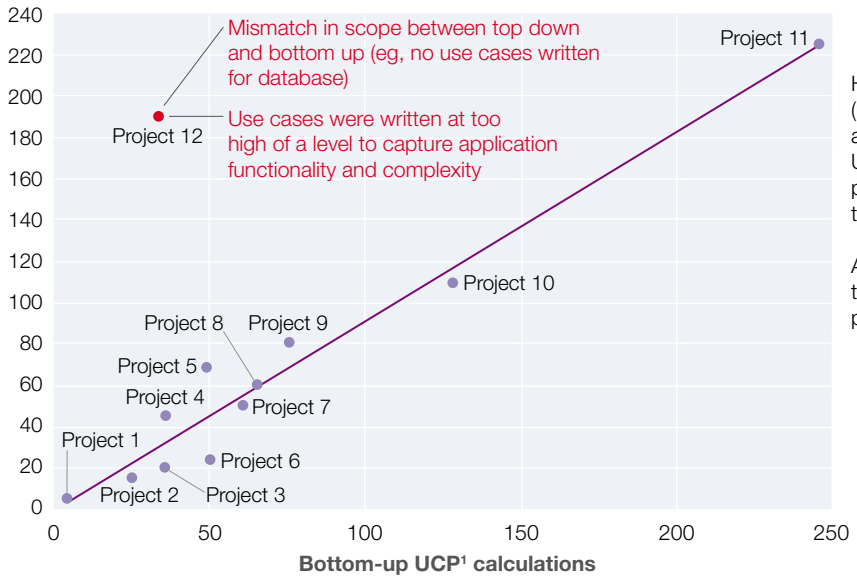
[4]The waterfall model is a sequential software-development process in which progress is seen as flowing steadily downward—like a waterfall—through the phases of initiation, analysis (requirements gathering), design, build, testing, release, and maintenance.

[5]In the agile approach, small teams work in weekly or biweekly sprint iterations, first delivering prototypes and then working with code that can be tried and tested.

Exhibit

Use-case-point calculations were aligned with the company's top-down assessment of the software functionality delivered.

Company-leadership estimation of software functionality delivered (index of effort project should have required)



High level of correlation (>80%) between top-down assessment and bottom-up UCP[1] analysis for nearly all projects (one is clear outlier that is explainable)

Analysis therefore supports the conclusion that UCPs[1] have predictive power

[1]Use-case point.

UCPs, how the organization will ensure that everyone is following the standard process, and how the metrics will be displayed and discussed.

Once the new design is complete, the pilot teams will train with the new processes and tools. Pilot teams can use previously completed projects to practice creating UCs and calculating UCPs. From there, the organization runs a pilot on actual projects to refine the processes and tools while addressing any gaps in the design. After completion of the pilot, organizations usually roll out UCs and UCPs more broadly in waves across the organization.

Throughout this process, it is critical to communicate a compelling change story. For example, the pilot team will need to explain the benefits of use cases to the business units, which naturally will be sensitive to any changes in the way requirements are gathered. Perhaps more important, there will likely be some resistance from within the development teams, whose members may not enjoy having their productivity measured.

What is critical for the ultimate acceptance of UCPs is how the leadership uses them. Developers will understand the rationale for using metrics to identify projects that are at risk of going off track. They will also understand the benefits of more accurately determining resources and timelines for projects, without over- or underscoping functional requirements. There is little that is more frustrating to application-development teams than pulling all-nighters to deliver what the business doesn't want or doesn't need, and then having to redo much

Pros and cons of four output metrics

Some organizations have adopted methods that measure the output of application development. We analyzed the performance of the four most widely used metrics, focusing on credibility, applicability, ease of use, usefulness, and scalability (exhibit).

Lines of code in an application have long been counted by organizations as a proxy for output. The total number of lines of code is easy to calculate, is applicable to nearly all types of application development, and is easily scaled across an organization. However, lines of code do not measure output—that is, the delivery of functionality required by the business. Moreover, because lines of code can only be calculated once code has been written, they are not useful for project planning or in-flight performance evaluation. Thus, other than as a very rough rule of thumb, lines of code are of limited use as an output metric.

Function points (FPs) rely on in-depth analysis of the functional and technical requirements of an application and therefore offer a way to measure output. The analysis involves a host of elements ranging from a count of transactions and files required to deliver the desired functionality to adjustment for the complexity of the project's technical requirements. As a result of this in-depth analysis, FPs are particularly useful in retrospective performance analysis, such as in determining whether an external vendor has met its contractual obligations.

However, FPs are difficult to calculate and require dedicated resources to measure and track. For this same reason, they are difficult to scale across an organization. Furthermore, FPs are less useful for project planning and in-flight performance evaluations because the application design and much of the build phase must be completed before FPs can be calculated. Finally, FPs are not well suited for the weekly or biweekly sprint iterations of agile software development.

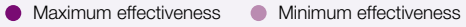
Story points (SPs) have gained considerable traction with teams following the agile methodology. SPs are an experience-based method that estimates the amount of software functionality based on user stories, or high-level descriptions of the functionality to be developed. This "gut feel" approach—with developers collectively scoring each requirement based on their prior experience—is both the strength and weakness of SPs. On the one hand, SPs are easy to calculate, can be taught quickly to multiple teams across an organization, and apply to any application-development project. SPs can also provide a rough order of magnitude for planning purposes and can be used by teams to track their progress. Therefore, SPs work well for a team following the agile approach of frequent iterations, where the primary purpose of SPs is to allocate the workload across the team.

However, because SPs are based solely on gut feel, they are too subjective or too easy to game to compare different development teams or even the performance of a single team over multiple periods.

Use-case points (UCPs) represent a sweet spot between FPs and SPs. UCPs are easier to calculate than FPs, provide a similar level of accuracy and objectivity, and require far less overhead. At the same time, UCPs provide significantly more accuracy and objectivity than SPs, without unduly adding overhead.

Exhibit

Organizations should assess metrics based on management objectives.



Management objective		LOCs[1]	SPs[2]	UCPs[3]	FPs[4]
Credibility	Accurate measurement of effort and complexity across teams and organizations				
Applicability	Can be leveraged for high portion of application-development portfolio				
Ease of use	Minimal overhead in calculating				
	Easy to scale out to organization (eg, training)				
	Consistent with both agile and waterfall methodologies				
Usefulness	Project planning and budgeting				
	In-flight team management				
	Retrospective performance review across teams and organizations				
	Vendor management				
Scalability	Ease of rollout across the organization				

[1]Lines of code.
[2]Story points.
[3]Use-case points.
[4]Function points.

of their hard work. If, however, UCPs are used merely as a means of rewarding or penalizing application developers, there is a much higher probability that there will be serious resistance.

• • •

The journey toward integrating a more efficient and effective way of gathering application-development requirements with a reliable output metric is not without its difficulties. However, the rewards are well worth the effort in a world where application development is an important key to success for almost any large enterprise.

Michael Huskins is an associate principal in McKinsey's Silicon Valley office; **James Kaplan** and **Krish Krishnakanthan** are principals in the New York office.