An agile process has two primary actors. One is the development team, which commits to develop software for a specific small set of requirements in a limited amount of time, and then develops and tests this software. The other is overall project management, which hands off the requirements and appropriate components to the teams and interfaces with stakeholders such as customers. In a similar vein, metrics for an agile process should have two parts: one for the development team and one for project management.

This article outlines metrics for an agile process being used at Brooks Automation. The process uses lightweight metrics at the development team level, where the focus is on developing working code, and more heavyweight metrics at the project management level, where the focus is on delivering a quality release to the customer. The authors describe the process and carry out a goal-question-metric (GQM) analysis to determine the goals and questions for such a process. They then examine the specific metrics used in the process, identifying them as either team-related or project-management-related; compare their scope to that shown by the GQM analysis; and identify issues in the existing metrics. Approaches to rectify those issues are then described.

**Key words:** agile, information systems, information technology, metrics, process improvement, project management, software, software development

---

**INTRODUCTION**

As a primary supplier of Real Time Enterprise software to the world’s foremost high-technology manufacturing environments, Brooks Software, a division of Brooks Automation, has adopted agile principles and metrics into its continuous improvement culture. Recent development efforts have shown a need to frequently reassess requirements for the intended software product and, consequently, replan the project, leading to significant product redesign and refactoring. Brooks recognized that an agile development paradigm would allow these significant changes to occur robustly. With such a process, the evolutions could be done deliberately, planned, and under organizational control.

Brooks Software has had several successful commercial software product releases using the software product life cycle and agile development processes. In addition to increased market and customer responsiveness, improvements in...
on-time delivery (OTD) predictability were significant. Defect reduction has also seen a 50 percent year-over-year improvement.

This article outlines a procedure for establishing metrics and metrics processes for an empirical (agile) process, with emphasis on using this procedure to specify improved metrics. Since an empirical process implements changes based on observations taken as the process evolves, with agile processes such as extreme programming (XP) being a specific form of empirical process, metrics assume an important role in such processes. In the procedure the authors use a lightweight metrics process at the development team level, where the emphasis is on simple data collection with only brief analyses carried out, and a more heavyweight process at the project management level, where more substantial data collection and management is done and deeper and more sophisticated analyses are carried out.

**LITERATURE REVIEW**

Agile methods originated in the mid-1990s as a counterbalance to the more formal, plan-driven methods that developed in the decade or two previously. Beck (2000) covers these kinds of development, especially XP. Goldratt (1992) discusses lean methods and theory of constraints, which also began to be applied to software life cycles and methods.

Boehm and Turner (2004) present criteria to determine when agile development is preferable and when plan-driven development is preferable. Manhart and Schneider (2004) carried out a case study using goal-question-metric (GQM) to select certain agile principles (automated unit test, test first) to combine with more traditional principles (documentation, coding rules) to implement in an embedded-software environment. Jalote et al. (2004) discuss the concept of timeboxing: the breaking up of a large project into small “timeboxes” that can then be carried out in separate steps (pipelined execution).

Williams et al. (2004) have developed a theoretical framework to evaluate XP and have applied it to an IBM-based case study. Cohn and Ford (2003) address the organizational aspects of the transition from a plan-driven to an agile process.

Heimann et al. (2005) identify three agile software development efforts described in the literature (Wood and Kleb (2003); an implementation of agility in a scientific software environment at NASA-Langley; Poole and Huisman (2001), an implementation in a maintenance environment at Iona; and Blotner (2002), an implementation in a start-up environment at Sabrix) and describe how the bipartite metrics approach would work for each case. In this article the authors take the approach in the aforementioned paper, extend and develop it further, and implement it in an ongoing environment, that is, the Brooks Agile Development Process (ADP).

Cao et al. (2004) describe the need for overlaying agile principles on top of XP practices as the development methodology to work effectively in a large-scale and complex software product development.

In *Software Quality Professional*, Gatlin (2003) discusses a case where a metrics implementation ran into difficulties, examines the lessons learned, and mentions common metrics and the weaknesses they can run into. Cross (2004) mentions various statistical pitfalls to watch out for in using software metrics, including confusions between discrete and continuous data and between accuracy and precision. Westfall (2006) outlines 12 steps in selecting, designing, and implementing software metrics to ensure understanding of their definition and purpose.

**METRICS IMPLEMENTATION CONSIDERATIONS FOR AGILE PROCESSES**

**Empirical Nature of the Agile Process and Resulting Metrics**

Plan-driven processes define the steps and procedures in advance for the development work to follow. Requirements are also assumed to be defined in advance and unchanging thereafter, except in special cases. There are fixed artifacts to produce, detailed documentation to produce and follow, and tests to be conducted after development is complete.

The agile development process, on the other hand, is more empirical. While some procedures are defined in advance, many are formulated as development proceeds, and even for predefined procedures, plenty of opportunity exists for flexibility and iteration. There is frequent opportunity to reaccess, redesign, refactor, and replan. Plans for any stage of a project are vaguely
specified at first and gradually become more detailed as the stage approaches, becoming specific only when the stage is actually reached.

Metrics for an agile process should reflect this empirical nature. Comparison of work done against work planned must adjust to a moving rather than a fixed “work planned” target. Measures of testing accomplished must similarly reflect that test cases planned will rapidly change. Defect counts must take into effect the amount of change taking place and the resulting complexity in order to be properly evaluated. Resource allocation for a given stage must fit into wide estimation intervals at first, narrowing down only as the stage approaches actuality.

Bipartite Nature of Metrics—Within/Among Agile Development Teams

The core of the agile software development effort is a development team—a small group of people who operate for a limited time to carry out a specific body of work. The teams are coordinated through an overall project management function, which parcels out work to the teams and integrates their work products into an overall release for customer delivery.

This leads to a dichotomy of interests between the teams and project management. Much of the information needed by project management is generated during the detail-oriented product generation effort by the teams. The teams, however, do not want to capture this information if it adversely affects their ability to complete their assignments in a timely fashion.

What is necessary, therefore, is to provide an approach where a team can generate and capture the necessary metric data as automatically as possible in developing its work products while enabling project management to access the resulting data for their more complex needs.

Therefore, the best way to organize metrics is as a bipartite collection, with one set of metrics focused on the development teams and the other focused on project management. The team metrics should be brief, so as to minimize the reporting burden on the teams; focused, in order to provide the teams with readily usable information for their development tasks; and short-term and rapid responding, to coincide with the short life of the team’s iterative cycle. Project management metrics should be comprehensive, so as to encompass the project’s entire complexity; integrative, to help with amalgamating the results of the development teams into a unified product with many interfaces; and long-term, to enable tracking of many small development teams throughout the project’s life.

Figure 1 shows the relationships among the teams, coordination and management function, and metrics and databases (note that the team-generated metrics connect into a central database from which project management develops its metrics). Figure 2 shows the development process as it moves from project management to the development team and back again.

THE BROOKS AGILE DEVELOPMENT PROCESS

The ADP leverages the theory of constraints concept, which advocates keeping each development cycle short. If one considers that the set of requirements committed and under construction is work-in-progress (WIP), the concept of the theory of constraints is to keep WIP backlog low, working on the highest priority requirements first. The development activity continues on the highest priority requirement until the product engineering team, which plays the role of a customer proxy, satisfactorily validates it from the end-user perspective. The aim is to complete work as quickly as possible.

The goals of the product release, captured in a project contract, are decomposed into manageable iterations through elaboration. Requirements
A Bipartite Empirically Oriented Metrics Process for Agile Software Development

FIGURE 2  Conceptual view: Agile development process (ADP)

![Diagram showing Agile development process (ADP)]

A product release consists of multiple features, and each feature is associated with a priority (see Figure 4). For example, in the case of \( F_2 P_2 \), \( F_2 \) represents feature no. 2 and \( P_2 \) is the priority level of feature no. 2. Each feature \( (F_i P_i) \) is broken down into one or more stories such that each story can be completed within a two-week construction cycle.

Stories represent the fundamental tracking unit for the WIP. By limiting construction to a two-week construction cycle within an overall six-week development cycle, the stories are “batched” for each iteration cycle and can be effectively managed. The iterations are organized as shown in Figure 5. Each six-week development cycle is completed in two iteration cycles, with the first cycle being used for planning and construction (development) and the second being used for development validation, verification (testing), and advance planning for the next iteration.

On average, seven to eight iterations are sufficient to provide overall functionality for a release, with two to three of these iterations reserved for backend bug fixes, final testing, and packaging. All the iterations are three calendar weeks long, with activities for previous and current iterations taking place in parallel. Hence, a typical release is about 24 calendar weeks long. Once feature freeze is reached, iteration cycles drop down to weekly and/or daily product build, and this is followed through the final testing and packaging phases of the development process.

FIGURE 3  Snippet of a sample story

High-Level Story Definition

Users will double click the Application Launcher icon on their desktop, or access it through the Windows Start menu. They will be presented with the standard login dialog for the aforementioned Application. If the user is able to log in successfully, the Application Launcher application will open and present the list of subapplications that are available for that user. The user may click and select any of the subapplications from the list; this action should invoke the subapplication. The user will NOT be required to log in again as long as the Application Launcher is still running.

If the user closes the Application Launcher, they will be required to log in again the next time they open the Application Launcher. The user name and password will NOT be stored locally on the client machine.

are specified in terms of stories, which encapsulate functionality definition in a form that can be easily communicated, validated with the customer and related stakeholders, and implemented in no more than two person-weeks of development effort. An example story is provided in Figure 3.
Story elaboration and iteration allow flexibility to schedule specific stories for each iteration cycle based on business priority, technical risk, and the need to include learning cycles and refactoring turns. Figure 2 summarizes this process.

ADP METRICS GOALS AND QUESTIONS

In keeping with the GQM paradigm (Basili, Caldiera, and Rombach 1994), the authors identify four goals for the metrics process, with a number of questions and consequent metrics for each goal.

The ADP process is guided by the following metrics goals: 1) projected project completion against targets; 2) quality of completed work; 3) ability to change content or priorities to address current business needs; and 4) ability to merge iteration-cycle products into a seamless overall release.

Each of these goals gives rise to a number of metrics questions:

1. Project-completion questions
   - What stories has the team completed?
   - How is the team doing compared to its task commitments?

2. Level-of-quality questions
   - How well does the team product from the current iteration fulfill the current requirements?
   - How well does the overall product fulfill the current requirements?
   - For a given team, how well is its developed product passing the quality assurance (QA) tests specific to its product?

FIGURE 4 High-level view: Release—Features—Stories

FIGURE 5 Three-week ADP iterative cycle for the ADP view
• Overall, how well is the completed work passing overall integration/system tests?
• Does the developed product possess systemic integrity and fitness for customer demonstration and eventual delivery?

3. Ability-to-change questions
• How have requirements changed thus far over the release lifetime?
• How are the current requirements matching up to customer (or market) expectations?

4. Ability-to-integrate questions
• Does each team know the inputs from and outputs to other product components that they need to address during their iteration?
• Have all the interactions among the various components and the various teams been accounted for, both in development planning and in integration and system testing?
• Does system testing cover all the code, interactions, and functionality?

ADP METRICS
Following are the important metrics that were selected to measure and track within the ADP process. While each can address multiple goals, the metrics are shown with the primary goals they address.

For each metric the authors identify whether the metric is calculated by the development team (such as the defects-found-fixed or story points metrics) or by project management (such as on-time delivery or velocity metrics). The authors also identify whether the metric is a traditional metric or developed to fit within an agile process such as ADP.

The use and consequent benefits of metrics within ADP is continuing to evolve. The authors’ plan is to report additional metrics-related information in future publications as the metrics evolve and more insights can be drawn from them.

Metrics for Goal 1: Projected Completion
To answer the question, “Are we on schedule?” an estimate of effort, anticipated productivity, and workload must be done and a schedule created. To carry out the assessment of productivity and workload necessary for this, one must measure the “mass” of a story. Brooks measures this by the “story point.” This is an agile-related measure and is calculated by the software development team in conjunction with the project management team. A story has a certain number of story points, obtained by multiplying a measure of the story’s size by a measure of its risk. As shown in Figure 6, the size of a story is given by a category of the effort needed for its development. A size can vary from a value of 1 through 10. The risk of a story is given by a category of its difficulty due to such factors as the novelty of a story or the lack of experience of its resources (a resource is a talent in such areas as development, documentation, verification, and so on). A risk can vary from a value of 1 to 3. Therefore, a story can have a value between 1 to 30 story points. Stories for all the iterations are tracked to provide a clear visibility of the current status to both the development and project management teams.

For the overall release at a project management level a key delivery metric is the organization’s ability to meet OTD to customer expectations. This is a traditional measure and is calculated by project management. This measurement tracks actual delivery against committed delivery within a given tolerance, usually 5 percent, with durations being measured in calendar workdays (see Appendix). During the design and planning phase the engineering team establishes the transition of a target date to a commit date. For ADP projects, past velocity rates and the complexity level of the current project are used to extrapolate project completion dates.

FIGURE 6 Guidelines for determining story size and risk

<table>
<thead>
<tr>
<th>ADP feature analysis—Size &amp; risk estimation table</th>
<th>Size legend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-2 days effort</td>
</tr>
<tr>
<td>2</td>
<td>3-5 days effort</td>
</tr>
<tr>
<td>3</td>
<td>6-10 days effort</td>
</tr>
<tr>
<td>4</td>
<td>11-15 days effort</td>
</tr>
<tr>
<td>5</td>
<td>16-20 days effort</td>
</tr>
<tr>
<td>6</td>
<td>21-25 days effort</td>
</tr>
<tr>
<td>7</td>
<td>26-30 days effort</td>
</tr>
<tr>
<td>8</td>
<td>31-35 days effort</td>
</tr>
<tr>
<td>9</td>
<td>36-40 days effort</td>
</tr>
<tr>
<td>10</td>
<td>41-45 days effort</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Risk legend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Low—No unknowns, have prior experience</td>
</tr>
<tr>
<td>2 Medium—Well understood, limited unknowns</td>
</tr>
<tr>
<td>3 High—No experience, high unknowns</td>
</tr>
</tbody>
</table>
To continually forecast and refine estimates a key productivity metric of the ADP is the velocity rate. This is an agile measure and is calculated by project management. The velocity is the rate at which story points are being completed per assigned capacity of resource (see Appendix).

Velocity measures the effectiveness of the project team to deliver functional units, as defined by story points. Measurement of the actual number of story points that can be produced by a project team of a given level of resource capacity, that is, velocity, enables management to assess team effectiveness. Hence, it ensures a balanced team in the project roles and allows monitoring of the process using theory of constraint principles for optimization as well as the basis for forward estimation using historical data. The unit for velocity is “story points generated per resource capacity.” For example, if four developers generate 24 story points in three weeks, the velocity is two story points per labor-week. Note that the resource capacity can vary from one iteration to another.

Figure 7 shows a sample velocity calculation over a six-iteration period, with both immediate and cumulative velocities shown in the accompanying graph.

A constituent element of velocity is the complexity factor. This is an agile-related measure and is calculated by project management. The complexity factor is a measurement of risk (see Figure 8 and Appendix).

The complexity factor shows the contribution of the risk within the total sizing of story points. It is the reciprocal of the size-weighted risk, and is used to compare the contribution of risk between modules of similar story-point values. Within modules of similar story points, those with low complexity factors have higher risks and lower sizes, while those with high complexity factors have lower risks and higher sizes, so that those with low complexity factors tend to be deserving of special scrutiny. The complexity factor also addresses Goal 1: Projected project completion, as

![Figure 7 Sample ADP velocity monitor](image)

**TABLE 8** Sample feature analysis, story points, and complexity factor calculation

<table>
<thead>
<tr>
<th>Feature ID</th>
<th>Feature Priority</th>
<th>High-level feature requirement description</th>
<th>Size $S_3$</th>
<th>Risk $R_3$</th>
<th>Points $S_3 \times R_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Name_1</td>
<td>1</td>
<td>Upgrade product to utilize Microsoft .Net 2.0 framework recommended by Microsoft and contains many critical bug fixes and enhancements. .Net 2.0 migration is required to use Visual Studio 2005.</td>
<td>8</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>Project Name_2</td>
<td>2</td>
<td>Define the details to support versioning for business logic component. Detailed design will be developed based on the outcome of the prototype. Versioning includes document states such as active, pending, archived, and off-line.</td>
<td>5</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>Project Name_3</td>
<td>3</td>
<td>Support dynamic Web service configuration for external interfaces. Adopt to current Web service standards.</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Project Name_4</td>
<td>4</td>
<td>Add visualization to view various different types of activities in a Pareto type chart.</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Project Name_5</td>
<td>5</td>
<td>Incorporate system user authentication and security model into the login application.</td>
<td>7</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>Project Name_6</td>
<td>6</td>
<td>Define the requirements to support application support for multiple user roles.</td>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>28</strong></td>
<td><strong>13</strong></td>
<td><strong>65</strong></td>
</tr>
</tbody>
</table>

Complexity factor: Sum of size/Sum of points 0.43
well as indirectly addressing Goal 2: Level of quality of completed work.

Velocity rates may change over time from project to project for similar components. The complexity factor helps identify the contribution of the risk aspect, with respect to the corresponding velocity trends. Figure 8 shows a sample calculation of the complexity factor.

**Issues for Goal 1:**

**Projected Completion**

Contending with moving planned-production targets

Velocity shows the production of stories, which can be used to compare actual production against planned production. However, in an agile development process “planned” is not a static unchanging benchmark against which one can compare production. As requirements change, features change, and, consequently, stories continue to be created and to evolve. To keep the “planned” picture up to date, one needs to track the stories in real time, adjusting them and the underlying requirements as they change, as well as estimate and forecast them into the future based on past and current customer volatility. Such a level of story creation ahead of time, which may not be needed, is going against the principles of theory of constraints. Keeping the release cycles short and maintaining a robust customer proxy role within the project team can help mitigate this concern to a large extent. Note, however, that within an iteration the production target is unchanging, so that the development team has a fixed target on which to focus. This is the heart of the bipartite process, where the development teams are provided with a fixed “waterfall-type” environment, while project management exercises agility in responding and adjusting to changes in customer requirements.

**Interface between requirements (customer input) and product backlog**

Agile processes need a requirements-management system, ironically more than do plan-based processes. This is especially true to link changes in requirements to maintenance of the product backlog list used to assign stories to teams in the iterations. This is also true in order to properly evaluate the impact of requirements changes and additions, so that the ongoing customer presence can be properly informed and give their proper input to the agile process. Within the agile development process this is performed by a customer proxy role that continually manages the requirements with the agile project team. Additionally, one of the primary functions of the customer proxy role is to validate the developed product from all the iterations for its usability from an end-user perspective. Developed product (stories) can be refactored to ensure end-user usability criteria are met.

**Metrics for Goal 2:**

**Product Quality**

To address the quality of the completed work metrics includes measuring tests run for each build including confidence levels over time for test coverage. Using Brooks Software’s Temporal reporting technologies against ClearQuest unified change management (UCM) repositories allows project teams and management to visualize these defect rates and found-fixed ratio. Both of these are traditional measures and are calculated by the team. The software quality assurance team supporting the project collects the underlying data.

A quantity used to evaluate test quality is a *weighted quality percentage* (see Figure 9 and Appendix). This is an agile-related measure, since it takes into account the age of the build on which a test was last run, and is calculated by the team using test-case weightings assigned by QA and project management based on the highest risk to the project deliverables. The range of the quality percentage is from –100 percent to 100 percent, where –100 percent is complete failure of the full suite of tests on the latest build and 100 percent is complete success of the full suite of tests on the latest build.

Each test case in a build is assigned a level from 1 to 4, with level 1 being the highest, and each level is assigned a point value. The considerations for the weightings are based on the criticality of the test. For example, a level 1 test may be a server component where a defect would have severe consequences, while a level 4 test may be a client on a function that has low frequency of use. The point values of failed test cases are subtracted from those of passed cases to obtain an adjusted point value. These values are divided by the total point value of the test cases to obtain the weighted quality percentage for
A Bipartite Empirically Oriented Metrics Process for Agile Software Development

A given build. In the example of Figure 9, the weighted quality percentage is 75 percent.

As the build frequency increases and the full suite of tests grows, quick turnaround time and cost considerations may not allow for the full suite of tests to be run for each build. One, therefore, must use test results from earlier builds, with a resulting loss in confidence as to whether those results are still appropriate. To take this “aging” into account the authors have established the “weighted quality percentage with confidence loss.” This is obtained by multiplying the adjusted point value for each test case by the timeliness value of the most recent build for which the test case was run. This timeliness value of each test case is 100 percent for the latest build. The same timeliness value for each test case will proportionally drop for subsequent future builds (for example, if the test case was last run on the seventh build and the current build is the 10th, the timeliness value is 70 percent). The metric provides a gauge for quality and project management on the potential risk of a regression to a code change not explicitly covered in the tests executed for the build. The weighted quality percentage with confidence loss is the sum of the timeliness-adjusted point values for all test cases divided by the total point value of all test cases. In the example of Figure 9, the weighted quality percentage with confidence loss is 52 percent.

Note that with the weighted quality percentage with confidence loss the negative impact of a failed test on the metric value lessens as the build on which the test occurs recedes further into the past without a retest taking place. It is therefore important that QA retest a failed test as soon as possible, especially for level 1 tests.

Weighted quality percentages (with or without confidence loss) may be very low in the first builds when tests tend to fail more often. By release time, however, they must be near 100 percent in order to comply with product release readiness policies, with explicit explanations of any workarounds agreed to by product stakeholders and management.

A metric used for product after shipment to the customer is voice of customer (VOC). This is an agile-related measure, tracked by the marketing and customer support groups and maintained by project management. The VOC metrics are grades of 1 to 5, with 5 representing the best answer, based on survey results calculated on a spreadsheet (see Figure 10) for the suite of products and versions delivered to the particular customer. To increase the quantitative sophistication of this metric, since it represents a key customer-satisfaction value, and to prevent an individual low score from being ignored because of an overall high average, any response or any product with a 3 or less triggers a root cause follow-up with the customer. Questions used in VOC metrics include:

### FIGURE 9 Sample weighted quality percentage

<table>
<thead>
<tr>
<th>Story</th>
<th>Level</th>
<th>Value</th>
<th>Last test result</th>
<th>Raw score</th>
<th>Last build tested</th>
<th>Confidence weight</th>
<th>Weighted score</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>10</td>
<td>Pass</td>
<td>10</td>
<td>7</td>
<td>0.7</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>8</td>
<td>Pass</td>
<td>8</td>
<td>8</td>
<td>0.8</td>
<td>6.4</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>5</td>
<td>Void</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>1</td>
<td>Pass</td>
<td>1</td>
<td>8</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>5</td>
<td>Pass</td>
<td>5</td>
<td>2</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>8</td>
<td>Fail</td>
<td>-8</td>
<td>8</td>
<td>0.8</td>
<td>-6.4</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>5</td>
<td>Pass</td>
<td>5</td>
<td>9</td>
<td>0.9</td>
<td>4.5</td>
</tr>
<tr>
<td>H</td>
<td>4</td>
<td>1</td>
<td>Pass</td>
<td>1</td>
<td>5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>I</td>
<td>3</td>
<td>5</td>
<td>Pass</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>J</td>
<td>2</td>
<td>8</td>
<td>Pass</td>
<td>8</td>
<td>5</td>
<td>0.5</td>
<td>4</td>
</tr>
<tr>
<td>K</td>
<td>1</td>
<td>10</td>
<td>Pass</td>
<td>10</td>
<td>7</td>
<td>0.7</td>
<td>7</td>
</tr>
<tr>
<td>L</td>
<td>2</td>
<td>8</td>
<td>Pass</td>
<td>8</td>
<td>4</td>
<td>0.4</td>
<td>3.2</td>
</tr>
<tr>
<td>M</td>
<td>3</td>
<td>5</td>
<td>Pass</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>N</td>
<td>2</td>
<td>8</td>
<td>Pass</td>
<td>8</td>
<td>9</td>
<td>0.9</td>
<td>7.2</td>
</tr>
<tr>
<td>O</td>
<td>4</td>
<td>1</td>
<td>Pass</td>
<td>1</td>
<td>7</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>P</td>
<td>3</td>
<td>5</td>
<td>Fail</td>
<td>-5</td>
<td>7</td>
<td>0.7</td>
<td>-3.5</td>
</tr>
<tr>
<td>Q</td>
<td>2</td>
<td>8</td>
<td>Pass</td>
<td>8</td>
<td>5</td>
<td>0.5</td>
<td>4</td>
</tr>
<tr>
<td>R</td>
<td>1</td>
<td>10</td>
<td>Pass</td>
<td>10</td>
<td>7</td>
<td>0.7</td>
<td>7</td>
</tr>
<tr>
<td>S</td>
<td>3</td>
<td>5</td>
<td>Pass</td>
<td>5</td>
<td>9</td>
<td>0.9</td>
<td>4.5</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>10</td>
<td>Pass</td>
<td>10</td>
<td>7</td>
<td>0.7</td>
<td>7</td>
</tr>
</tbody>
</table>

Maximum possible score: 126
Weighted score: 95
Weighted score with confidence loss: 65

Weighted quality %: 75.4%
Weighted quality % with confidence loss: 51.5%
Automation of tests helps to reduce testing complexity by routinely carrying out straightforward tests such as regression or duration tests, allowing one to dedicate more resources to less routine tests.

Establishing metrics for code and functional coverage is a future need that will allow the understanding of how much test coverage exists for each build.

Metrics for Goal 3: Ability to Change Content or Priorities

The ADP process does not bind requirements to a release until the iteration planning activities. This allows flexibility to recast priorities for all the iteration. The cost of this flexibility is that it requires a dedicated effort and resources for ongoing product (requirements) engineering throughout the project life span. Each iteration has an iteration plan that defines the prioritized set of requirements (story points) to be addressed in each three-week cycle. Each cycle is planned based on current business conditions and is short enough that change is minimal. Requirements that do not get successfully implemented in the iteration are candidates for the next cycle based on overall priorities.

Currently, initial release plan stories and actual completed stories are tracked. Unlike traditional processes the planned stories vary over time as requirements change in keeping with the changing marketplace requirements. Therefore, once they are defined for implementation in a given iteration, actual stories are tracked. The difference between the initially planned story points and the actual developed stories is called the story point variance (see Appendix). This is an agile-related metric that is calculated by project management. The purpose of the variance between the initially planned stories and actual developed stories...
is to help determine a project management buffer for future releases. Knowledge of this historical variance is helpful to keep up with the desire to meet the OTD metric discussed earlier. Within a project ADP measures the actual stories completed for delivery to the customer.

**Issues for Goal 3: Ability to Change Content or Priorities**

Currently, the suite of project management metrics for goal 3 is not as rich as those for goals 1 and 2. Capturing requirements changes at the various levels would allow for better forecasting of how much the requirements can be expected to change over the project’s life. Story point variance is currently being employed in this regard. Other metrics under consideration include:

- **Market changes**: The market requirements are allocated to a project contract that defines the high-level targets for each release. By exercising version and configuration control on the project contract one can monitor its evolution over time due to market changes and consequent changes in requirements.

- **Changes in the project backlog**: Within the iteration planning, the project backlog is the source of stories to be developed within each cycle. Within an iteration, the two-to-three week development cycle allows the iteration to have its own planning process to allow for adaptation and change. Through change control on the project backlog over the various iterations, one can keep track of changes in the project backlog as the iterations are carried out. While market changes and story point variance focus on the inflow to the project backlog, this metric focuses on the outflow from the backlog. The project backlog and its administration is a core concept of the ADP and allows for a high level of agility and change control.

These metrics are agile-related and can be maintained by project management. They may require somewhat more of a workload than the current story-tracking metric from the project management team. Automated database systems can help in this regard, particularly a requirement-traceability tool and a good software configuration management (SCM) tool.

**Metrics for Goal 4: Ability to Merge Iteration Cycles Into a Seamless Overall Release**

For a commercial application supplier like Brooks Software it is important that software releases produced with an agile process also include whole product readiness that includes integration with other software and hardware components such as operating system and infrastructure configurations, documentation, installation, upgrade, and other elements that meet customer expectations and maximizes user experience.

Overall shipment readiness is determined by the engineering and launch readiness process that ensures all product elements are available to ship through measurement of out-of-box quality (OBQ), which includes the measurement of post-release customer-reported defects (see Figure 11 and Appendix). This is a traditional measure and is calculated by project management. It measures overall product quality as experienced by the customer. This measures quality beyond code-defect rates and ensures whole product packaging objectives are being met. Figure 11 shows an example OBQ graph. The bar graph shows that installation and configuration defects occur most frequently and the cumulative line graph shows that the degree of imbalance is approximately 40/60 (that is, the top 40 percent of the categories accounts for 60 percent of the total defects).

**FIGURE 11** Sample OBQ measure
Issues for Goal 4: Ability to Merge Iteration Cycles Into a Seamless Overall Release

This is an area that is difficult to establish quantitative metrics for and uses more of a peer and stakeholder review process to ensure product readiness. This is addressed in the area of customer acceptance testing and installation and delivery sign-off. Earlier in the development cycle other processes can be developed to address the various questions in this goal area:

- Does each team know the inputs from and outputs to other product components that they need to address during their iteration? This can be addressed through peer review of the team’s plan for story development and tracked by various input-output analyses among stories. Traceability and configuration management tools can be useful here.

- Have all the interactions among the various components and the various teams been accounted for, both in development planning and in integration and system testing? This can be similarly addressed through peer reviews of the development plans and the system-test plans and can be similarly tracked by input-output and traceability analyses.

- Does system-testing cover all the code, interactions, and functionality? Currently, the full set of stories is tested. This is addressed in system test planning. In addition, the authors are considering adding metrics that quantify coverage of code and functions. This will help detect defects relating to unplanned functionality in the code that are not fully captured in the stories and design specifications, as well as “dead code,” that is, hard-to-reach portions of the code that may not be covered unless the test effort specifically focuses on that part of the code.

The authors are also looking to increase the level of sophistication in this goal area, as well as to develop a number of specific measures.

Next Steps

Future papers will discuss the advances of these areas:

1. Additional bipartite team and project metrics for process improvement
2. Further sophistication in the data collection system for the bipartite agile metrics
3. Further implementation of process improvements incorporating the bipartite approach

Another area of longer-term interest for Brooks is scaling the agile process to support distributed teams. With advances in software engineering environments and with the advent of collaboration systems from suppliers like Microsoft, IBM, and Oracle, the benefits of the ADP agile process may be able to be scaled to benefit disbursed global teams.

SUMMARY AND CONCLUSIONS

An agile process can be established that optimizes around short planning cycles and theory of constraints. It is consistent with proven advances in other domains such as product life-cycle management and manufacturing in which the discipline to remove waste from all processes has proven as required for sustaining growth. For commercial software companies where time-to-market, building it right, and cost pressures drive management investment decisions, agile has moved from being a fad to being a trend.

The authors have established a bipartite suite of metrics, with some, such as defect rates, defects found and fixed, and weighted quality percentage, oriented to the development team, others, such as velocity, complexity factor, OTD, VOC, and story point variance, oriented to project management, and others, in particular story points, oriented to both the development team and project management. Many of these metrics, such as story points, story point variance, velocity, complexity factor, weighted quality percentage, and VOC, have been created to address the specifics of an agile development process such as ADP.

In particular, the velocity metric has been instrumental in providing a meaningful way to predict the availability of future product version release time-frame. Velocity metric is used as a throughput rate, which is generally a measure of effort in the traditional sense. The differentiating factor of this metric is that velocity also captures a historical level of difficulty that is not captured in traditional throughput rates.
Hence, using velocity to determine future release timeframe has been more beneficial and accurate as compared to traditional throughput measures.

In addition, the complexity factor metric has provided valuable information about the level of difficulty for a given product version. Using historical complexity factor information an appropriate level of risk management planning can be put in place for a given product version. Paying too much or too little attention to various forms of risks can be a key ingredient for delayed software delivery and/or lack of needed level of quality in the released software product.

All of the metrics identified in this article have provided Brooks Software invaluable insights and measurable tools in effectively managing both the project and product portfolio that best meets the needs of its customers. Metrics and the associated historical information collected thus far have proved to be very helpful to both the project team and the management team. The ADP process and its associated metrics have evolved over a three-year period, and this journey will continue over time to adapt the process further, thereby advancing the discipline of software engineering in an agile environment.

REFERENCES


BIographies

David I. Heimann is a professor in the Management Science and Information Systems Department of the College of Management at the University of Massachusetts, Boston. He has a doctorate in computer science from Purdue University. He has held positions in government and industry, performing activities in reliability modeling, simulation, database management systems, probabilistic modeling, software analysis, and software process improvement. Among his publications are articles over several volumes of the “Proceedings of the Annual Reliability and Maintainability Symposium,” as well as a book-length article on system availability in “Advances in Computers.” He can be reached by e-mail at David.Heimann@umb.edu.

Peter R. Hennessey is director of central engineering programs for Brooks Software. He has more than 20 years of experience developing and delivering automation and business software to some of the worlds most advanced manufacturing technology companies. At Brooks Software, Hennessey works across the engineering community to establish product and engineering process, methods, and architectures to maintain Brook’s worldwide operational and product leadership. He has a master’s degree in information systems from Northeastern University and a bachelor’s degree in economics from the University of Massachusetts, Amherst.
Appendix

\[ \text{OTD} = \frac{(\text{actual available to ship date} - \text{planned commit date})}{\text{project duration}} \]

Metric: True or false: "Is OTD ≤ a given tolerance level?"

\[ \text{Velocity} = \frac{\text{Sum of actual story points}}{\text{Sum of capacity}} \]

\[ V = \frac{\sum S_i R_i}{\sum C_i} \]

where \( s \) is a story, \( S_s \) the size of story \( s \), \( R_s \) the risk for story \( s \), and \( C_i \) the capacity of resource \( i \)

Note that story points = Story size * Complexity risk

The complexity factor is defined by the following:

\[ C = \frac{\sum S_s R_s}{\sum S_s R_s} \]

where \( s \) is a story, \( S_s \) the size of story \( s \), and \( R_s \) is the risk for story \( s \)

Weighted quality percentage = \( \frac{(\text{test case values} * \text{test-result weighting})}{(\text{test case values})} \)

Weighted quality percentage with confidence loss = \( \frac{(\text{test case values} * \text{test-result weighting} * \text{build timeliness})}{(\text{test case values})} \)

\[ \text{Story point variance} = \frac{(P_s - A_s)}{P_s} \]

where \( P_s \) is the total initially planned story points and \( A_s \) is the total actual completed story points

\[ \text{OBQ} = \% \text{ of defect categories per month (installation, packaging, documentation, software, etc.)} \]