13.5 PRAGMATIC SOFTWARE METRICS

Measuring is useful, but it doesn't do any thinking for the decision makers. It only provides data to help them ask the right questions, understand the context, and make objective decisions. Because of the highly dynamic nature of software projects, these measures must be available at any time, tailorable to various subsets of the evolving product (release, version, component, class), and maintained so that trends can be assessed (first and second derivatives with respect to time). This situation has been achieved in practice only in projects where the metrics were maintained on-line as an automated by-product of the development/integration environment.

The basic characteristics of a good metric are as follows:

1. *It is considered meaningful by the customer, manager, and performer.* If any one of these stakeholders does not see the metric as meaningful, it will not be used. "The customer is always right" is a sales motto, not an engineering tenet. Customers come to software engineering providers because the providers are more expert than they are at developing and managing software. Customers will accept metrics that are demonstrated to be meaningful to the developer.
2. *It demonstrates quantifiable correlation between process perturbations and business performance.* The only real organizational goals and objectives are financial: cost reduction, revenue increase, and margin increase.
3. *It is objective and unambiguously defined.* Objectivity should translate into some form of numeric representation (such as numbers, percentages, ratios) as opposed to textual representations (such as excellent, good, fair, poor). Ambiguity is minimized through well-understood units of measurement (such as staff-month, SLOC, change, function point, class, scenario, requirement), which are surprisingly hard to define precisely in the software engineering world.
4. *It displays trends.* This is an important characteristic. Understanding the change in a metric's value with respect to time, subsequent projects, subsequent releases, and so forth is an extremely important perspective, especially for today's iterative development models. It is very rare that a given metric drives the appropriate action directly. More typically, a metric presents a perspective. It is up to the decision authority (manager, team, or other information processing entity) to interpret the metric and decide what action is necessary.
5. *It is a natural by-product of the process.* The metric does not introduce new artifacts or overhead activities; it is derived directly from the mainstream engineering and management workflows.

6. *It is supported by automation.* Experience has demonstrated that the most successful metrics are those that are collected and reported by automated tools, in part because software tools require rigorous definitions of the data they process.

When metrics expose a problem, it is important to get underneath all the symptoms and diagnose it. Metrics usually display effects; the causes require synthesis of multiple perspectives and reasoning. For example, reasoning is still required to interpret the following situations correctly:

- A low number of change requests to a software baseline may mean that the software is mature and error-free, or it may mean that the test team is on vacation.
- A software change order that has been open for a long time may mean that the problem was simple to diagnose and the solution required substantial rework, or it may mean that a problem was very time-consuming to diagnose and the solution required a simple change to a single line of code.
- A large increase in personnel in a given month may cause progress to increase proportionally if they are trained people who are productive from the outset. It may cause progress to decelerate if they are untrained new hires who demand extensive support from productive people to get up to speed.

Value judgments cannot be made by metrics; they must be left to smarter entities such as software project managers.

13.6 METRICS AUTOMATION

There are many opportunities to automate the project control activities of a software project. For managing against a plan, a software project control panel (SPCP) that maintains an on-line version of the status of evolving artifacts provides a key advantage. This concept was first recommended by the Airlie Software Council [Brown, 1996], using the metaphor of a project "dashboard." The idea is to provide a display panel that integrates data from multiple sources to show the current status of some aspect of the project. For example, the software project manager would want to see a display with overall project values, a test manager may want to see a display focused on metrics specific to an upcoming beta release, and development managers may be interested only in data concerning the subsystems and components for which they are responsible. The panel can support standard features such as warning lights, thresholds, variable scales, digital formats, and analog formats to present an overview of the

current situation. It can also provide extensive capability for detailed situation analysis. This automation support can improve management insight into progress and quality trends and improve the acceptance of metrics by the engineering team.

To implement a complete SPCP, it is necessary to define and develop the following:

1. Metrics primitives: indicators, trends, comparisons, and progressions
2. A graphical user interface: GUI support for a software project manager role and flexibility to support other roles
3. Metrics collection agents: data extraction from the environment tools that maintain the engineering notations for the various artifact sets
4. Metrics data management server: data management support for populating the metric displays of the GUI and storing the data extracted by the agents
5. Metrics definitions: actual metrics presentations for requirements progress (extracted from requirements set artifacts), design progress (extracted from design set artifacts), implementation progress (extracted from implementation set artifacts), assessment progress (extracted from deployment set artifacts), and other progress dimensions (extracted from manual sources, financial management systems, management artifacts, etc.)
6. Actors: typically, the monitor and the administrator

Specific monitors (called *roles*) include software project managers, software development team leads, software architects, and customers. For every role, there is a specific panel configuration and scope of data presented. Each role performs the same general use cases, but with a different focus.

- Monitor: defines panel layouts from existing mechanisms, graphical objects, and linkages to project data; queries data to be displayed at different levels of abstraction
- Administrator: installs the system; defines new mechanisms, graphical objects, and linkages; handles archiving functions; defines composition and decomposition structures for displaying multiple levels of abstraction

The whole display is called a panel. Within a panel are graphical objects, which are types of layouts (such as dials and bar charts) for information. Each graphical object displays a metric. A panel typically contains a number of graphical objects positioned in a particular geometric layout. A metric shown in a graphical object is labeled with the metric type, the summary level, and the instance name (such as lines of code, subsystem, server1). Metrics can be displayed in two modes: value, referring

eg - Sensex Chart in a Business News
Polling Panel in Election News

to a given point in time, or graph, referring to multiple and consecutive points in time. Only some of the display types are applicable to graph metrics.

Metrics can be displayed with or without control values. A control value is an existing expectation, either absolute or relative, that is used for comparison with a dynamically changing metric. For example, the plan for a given progress metric is a control value for comparing the actuals of that metric. Thresholds are another example of control values. Crossing a threshold may result in a state change that needs to be obvious to a user. Control values can be shown in the same graphical object as the corresponding metric, for visual comparison by the user.

Indicators may display data in formats that are binary (such as black and white), tertiary (such as red, yellow, and green), digital (integer or float), or some other enumerated type (a sequence of possible discrete values such as sun..sat, ready-aim-fire, jan..dec). Indicators also provide a mechanism that can be used to summarize a condition or circumstance associated with another metric, or relationships between metrics and their associated control values.

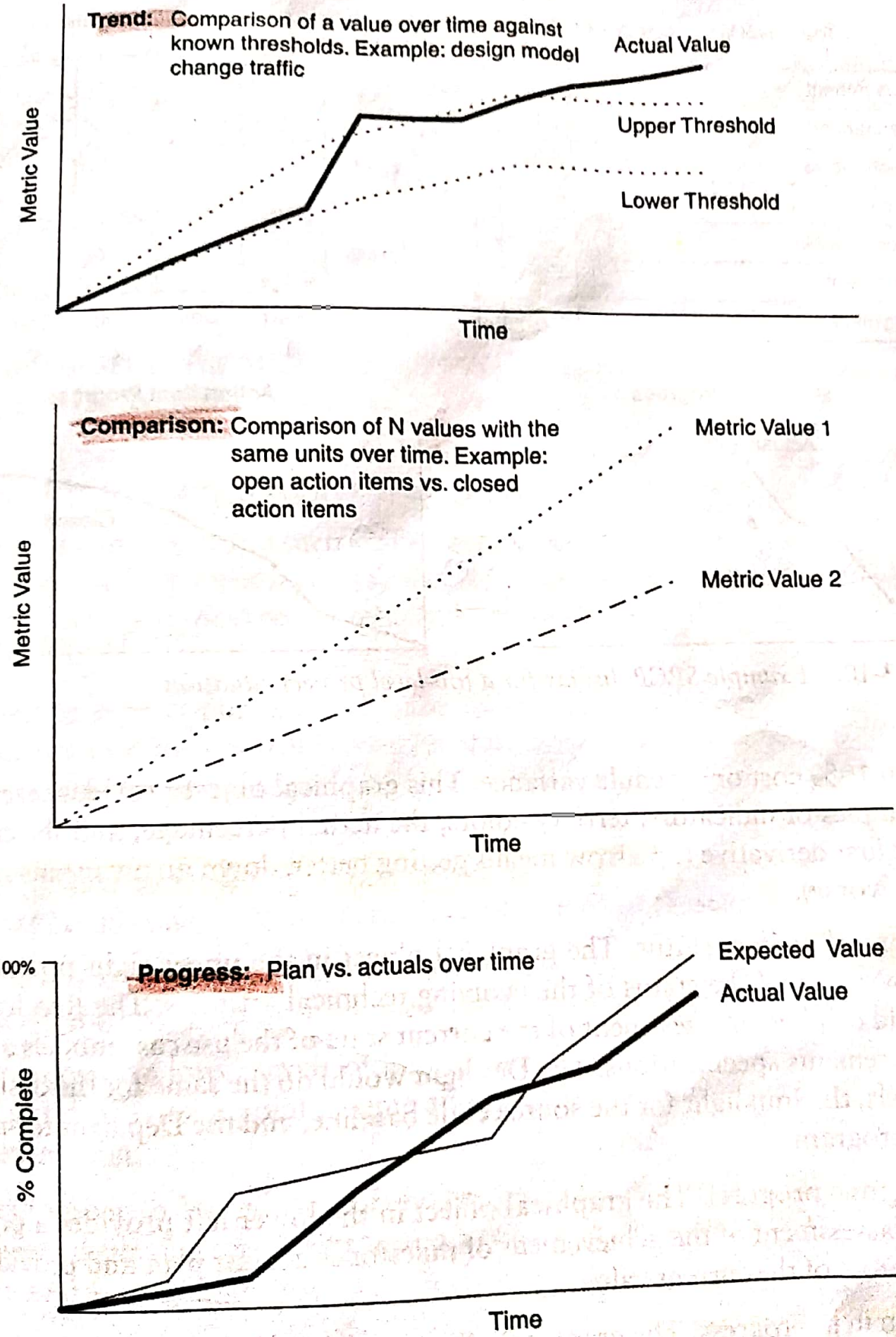
A trend graph presents values over time and permits upper and lower thresholds to be defined. Crossing a threshold could be linked to an associated indicator to depict a noticeable state change from green to red or vice versa. Trends support user-selected time increments (such as day, week, month, quarter, year). A comparison graph presents multiple values together, over time. Convergence or divergence among values may be linked to an indicator. A progression graph presents percent complete, where elements of progress are shown as transitions between states and an earned value is associated with each state. Trends, comparisons, and progressions are illustrated in Figure 13-9.

Metric information can be summarized following a user-defined, linear structure. (For example, lines of code can be summarized by unit, subsystem, and project.) The project is the top-level qualifier for all data belonging to a set (top-level context). Users can define summary structures for lower levels, select the display level based on previously defined structures, and drill down on a summarized number by seeing the lower level details.

Figure 13-10 illustrates a simple example of an SPCP for a project. In this case, the software project manager role has defined a top-level display with four graphical objects.

Work Breakdown Structure

1. Project activity status. The graphical object in the upper left provides an overview of the status of the top-level WBS elements. The seven elements could be coded red, yellow, and green to reflect the current earned value status. (In Figure 13-10, they are coded with white and shades of gray.) For example, green would represent *ahead of plan,* yellow would indicate *within 10% of plan,* and red would identify elements that have a greater



RE 13-9. *Examples of the fundamental metrics classes*

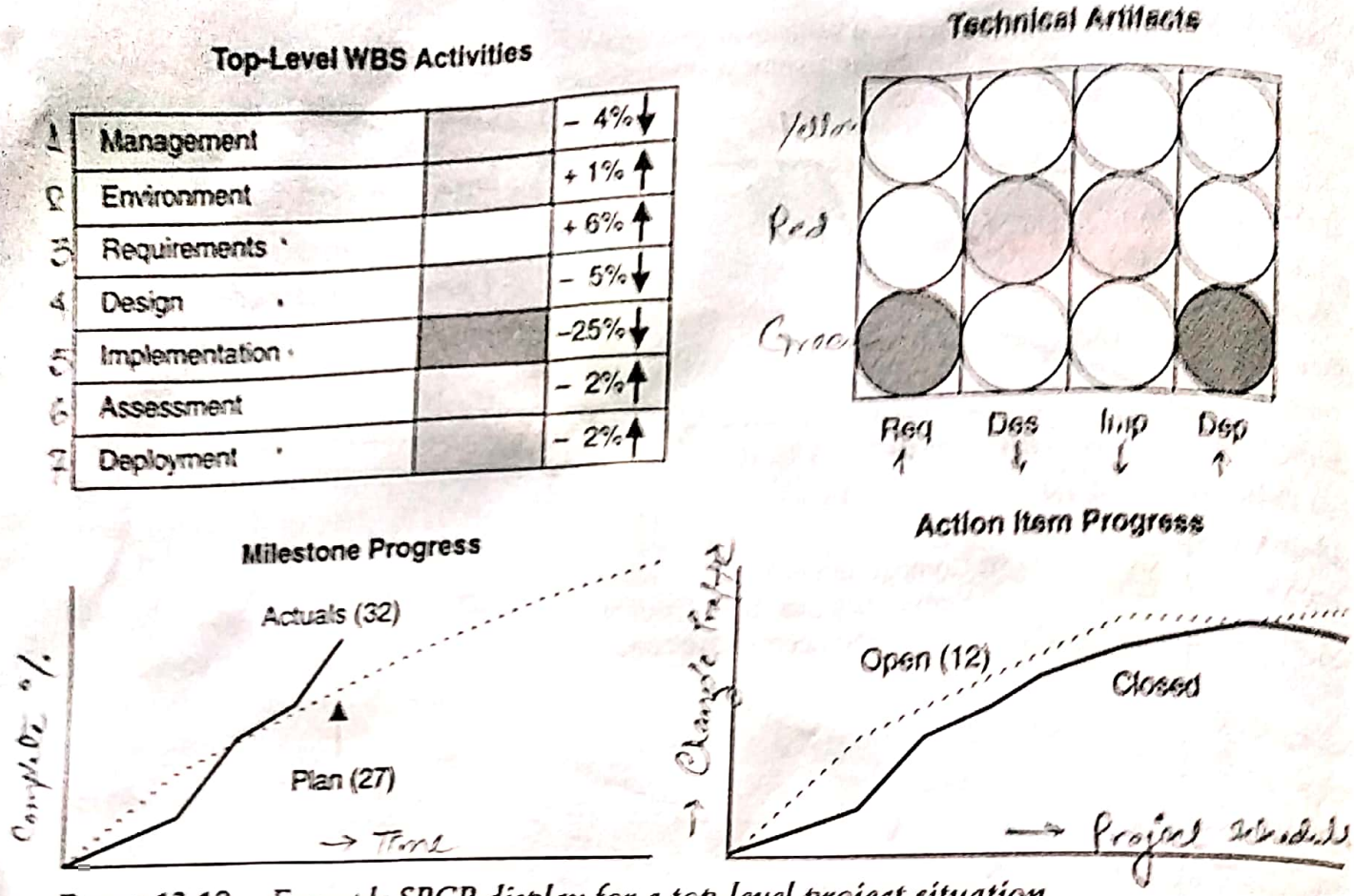


FIGURE 13-10. *Example SPCP display for a top-level project situation*

than 10% cost or schedule variance. This graphical object provides several examples of indicators: tertiary colors, the actual percentage, and the current first derivative (up arrow means getting better, down arrow means getting worse).

2. Technical artifact status. The graphical object in the upper right provides an overview of the status of the evolving technical artifacts. The Req light would display an assessment of the current state of the use case models and requirements specifications. The Des light would do the same for the design models, the Imp light for the source code baseline, and the Dep light for the test program.

3. Milestone progress. The graphical object in the lower left provides a progress assessment of the achievement of milestones against plan and provides indicators of the current values.

4. Action item progress. The graphical object in the lower right provides a different perspective of progress, showing the current number of open and closed issues.

Figure 13-10 is one example of a progress metric implementation. Although the example is trivial, it provides a view into the basic capability of an SPCP display. The