International Journal of Information Technology and Knowledge Management
July-December 2009, Volume 2, No. 2, pp. 277-281

ANALYSIS AND DESIGN OF CORE METRICS FOR MODERN SOFTWARE PROJECTS

K. P. Yadav* & Raghuraj Singh**

Size measurement methods have played a key role in helping to solve real-world problems of estimating, supplier/customer disputes, performance improvement and the management of outsourced contracts. We would like to say that estimating and managing a project's effort, staffing, schedule, cost, risks, quality and other factors is crucial. Yet all these are measures of input. Process management is enabled by feedback loops. So it is also necessary to measure the output from the software process. This starts with measurement of the size of the requirements. To solve a problem, it is necessary to measure its size, in order to assess the various solution options, calculate the relative costs, compare the benefits, before finally committing to one preferred approach.

Software Project / Engineering Metrics

Many different metrics may be of value in managing a modern process/projects. I have settled on EIGHT CORE METRICS that should be used on all types of modern software projects, in which four are management metrics and four are quality metrics. These may be called as indicators.

a) Management Metrics

- Work and progress
- Budgeted cost and expenditures
- Staffing and tem dynamics
- Software project control panel

b) Quality Metrics

- Change traffic and stability
- Breakage and modularity
- Rework and adaptability
- Mean time between failures (MTBF) and maturity

The following Table describes the core software metrics. Each metric has two dimensions: a static value used as an objective, and dynamic trend used to manage the achievement of that objective. While metrics values provide one dimension of insight, metrics trends provide a more important perspective for managing the process. Metrics

Table
Overview of the Eight Core Metrices

Metric	*Purpose*	*Perspectives*
Work and progress	Iteration planning, plan vs. actuals, management indicator	SLOC, function points, object points, scenarios, test cases, SCOs
Budgeted cost and expenditures	Financial insight, plan vs. actuals, management indicator	Cost per month, full time staff per month, percentage of budget expended
Staffing and term dynamics	Resource plan vs. actuals, hiring rate, attrition rate	People per month added, people per month leaving
Software project control panel	To show current status of the project	Overall project values, thresholds of projects
Change traffic and stability	Iteration planning, management indicator of Schedule convergence	SCOs opened vs. SCOs closed, by type (0,1,2,3,4), by release/ component/subsystem
Breakage and modularity	Convergence, software scrap, quality indicator	Reworked SCOs per change, by type (0, 1, 2, 3, 4), by release/component/subsystem
Rework and adaptability	Convergence, software reworked, quality indicator	Average hours per change by type (0,1,2,3,4), by release/component/subsystem
MTBF and maturity	Test coverage/adequacy, robustness for use, quality indicator	Failure counts, test hours until failure, by release/component/subsystem

* Dronacharya College of Engg., Greater Noida, U.P.
Email: karunesh732@yahoo.co.in

** Deptt. of CSE, H.B.T.I., Kanpur, U.P.
Email: raghurajsingh@rediffmail.com

trends with respect to time provide insight into how the process and product are evolving. Iterative development is about managing change, and measuring change is the most important aspect of the metrics program. Absolute values of productivity and quality improvement are secondary issue until the fundamental goal of management has been achieved: predictable cost and schedule performance for a given level of quality.

The eight core metrics can be used in numerous ways to help manage projects and organizations. In an iterative development project or an organization structured around a software line of business, the historical values of previous iterations and projects provides precedents data for planning subsequent iterations and projects. Consequently, once metrics collection is ingrained, a project or organization can improve its ability to predict the cost, schedule, or quality performance of future work activities.

The eight core metrics are based on common sense and field experience with both successful and unsuccessful metrics program. Their attributes include the following:

- They are simple, objective, easy to collect, easy to interpret, and hard to misinterpret.
- Collection can be automated and no intrusive.
- They provide for consistent assessments throughout the life cycle and are derived from the evolving product baselines rather than from a subjective assessment.
- They are useful to both management and engineering personnel for communicating progress and quality in a consistent format.
- Their fidelity improves across the life cycle.

Management Metrics

There are four fundamental sets of management metric: technical progress, financial status, staffing progress, and current project status/value. By examining these perspectives, management can generally assess whether a project is on budget and on schedule. Financial status is very well understood; it always has been. Most managers know their resource expenditures in terms of costs and schedule. The problem is to assess how much technical progress has been made. Conventional projects whose intermediate products were all paper documents relied on subjective assessments of technical progress or measured the number of documents competed. While these documents did reflect progress in expending energy, they were not very indicative of useful work being accomplished.

The management indicators not very recommended here include standard financial status primary on an earned value system, objective technical progress metrics tailored to the primary measurement criteria for each major team of the organization, and staffing metrics that provide insight into team dynamics.

Work and Progress

The various activities of an iterative development project can be measured by defining a planned estimate of the work in an objective measure, then tracking progress (work completed over time) against that plan. Each major organization team should have at least one primary progress that it is measured against. For the standard teams the default perspectives of this metric would be as follows.

- Software architectures team: use cases demonstrated
- Software development team: SLOC under baseline change management, SCOs closed.
- Software management team: milestones completed.

Budgeted Cost and Expenditures

To maintain management control, measuring cost expenditure over the project life cycle is always necessary. Through the judicial use of the metrics for work and progress, a much more objective assessment of technical progress can be performed to compare with cost expenditures. With an iterative development process, it is important to plan the near-term activities (usually a window of time less than six months) in detail and leave the far-term activities as rough estimates to be refined as the current iteration is winding down and planning for the next iteration becomes crucial.

Tracking financial progress usually takes on an organization-specific format. One common approach to financial performance measurement is use of an earned value system, which provides highly details cost and schedule insight. Its major weakness for software projects has traditionally been the inability to assess the technical progress (% complete) objectively and accurately. While this will always be the case in the engineering stage of a project, earned value systems have proved to be effective for the production stage, where there is high-fidelity tracking of actual versus plans predictable results. The other core metrics provide a framework for detailed and realistic quantifiable backup data to plan and track against, especially in the production stage of software, when the cost and schedule expenditures are highest.

Staffing and Team Dynamics

An iterative development should start with a small team until the risks in the requirements and architecture have been suitable resolved. Depending on the overlap of iterations and other project-specific circumstances, staffing can vary.

For discrete, one-of-a-kind development efforts (such as building a corporate information system), the staffing profile would be typical. It is reasonable to expect the maintenance team to be smaller than the development team for these sorts of development. For a commercial product development, the sizes of the maintenance and development teams may be the same. When long-lived, continuously improved products are involved, maintenance is just continuous construction of new better releases.

Tracking actual versus planned staffing is a necessary and well-understood management metric. There is one other important management indicator of changes in project momentum: the relationship between attrition and additions. Increases in staff can slow overall project progress as new people consume the productive time of existing people in coming up to speed. Low attribution of good people is a sign of success. Engineers are highly motivated by making progress in getting something to work; this is the recurring theme underlying an effect iterative development process. If this motivation is not there, good engineers will migrate elsewhere. An increase in unplanned attribution-namely, people leaving a project prematurely-is one of the most glaring indicators that a project that a project is destined for trouble. The causes of such attrition can vary, but they are usually personnel dissatisfaction with management methods, lacks of teamwork, or probability of failure in meeting the planned objectives.

Software Project Control Panel

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 a 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.

Quality Metrics

The four quality metrics are based primarily on the measurement of software change across evolving baselines of engineering data (such as design models and source code).

Change Traffic and Stability

Overall change traffic is one specific indicator of progress and quality. Change traffic is defined as the number of software change orders opened and closed over the life cycle. This metric can be collected by change type, by release, across all release, by team, by components, by subsystem, and so forth. Coupled with the work and progress metrics, it provides insight into the stability of the software and its convergence towards stability (or divergence towards instability). Stability is defined as the relationship between opened versus closed SCOs.

The next three quality metrics focus more on the quality of the product.

Breakage and Modularity

Breakage is defined as the average extent of change, which is the amount of software baseline that needs rework (in SLOC, function points, components, subsystems, files, etc.). Modularity is the average breakage trend over time. For a healthy project, the trend expectation is decreasing or stable.

This indicator provides insight into the benign or, malignant character of software change. In a mature interactive development process, earlier changes are expected to result in more scrap than later changes. Breakage trends that are increasing with time clearly indicate the product maintainability is suspect.

Rework and Adaptablity

Rework is defined as the average cost of change, which is efforts to analyze, resolve, and retest all charges to software baselines. Adaptability is defined as the rework trend over time. For a healthy project, the trend expectation is decreasing or stable.

Not all changes are created equal. Some changes can be made in a staff-hour, while others take staff-weeks. This metric provides insight into rework measurement. In a mature iterative development process, earlier changes (architectural changes, which affect multiple components and people) are expected to require more rework than later changes (Implementation changes, which tend to be confined to a single components or person). Rework trends that are increasing with time clearly indicates that product maintainability is suspect.

MTBF and Maturity

MTBF is the average usages time between software faults. In general terms, MTBF is computed by dividing the test hours by the number of type 0 and type 1 SCOs. Maturity is defined as the MTBF trend over time.

Early insight into maturity requires that an effective test infrastructure be established. Conventional testing approaches for monolithic software programs focused on achieving complete coverage of every line of code, every branch, and so forth. In today's distributed and

componentized software systems, such complete test coverage is achievable only for discrete components. Systems of components are more efficiently tested by using statistical techniques. Consequently, the maturity metrics measure statistics over usage time rather than product coverage.

Software errors can be categorized into two types: deterministic and nondeterministic. Physicist would characterize these as Bohr-bugs and Heisen-bugs, respectively. Bohr-bugs represent a class of errors that always results when the software is stimulated in a certain way. These errors are predominantly caused by coding errors, and changes are typically isolated to a single component. Heisen-bugs are software faults that are coincidental with a certain probabilistic occurrence of a given situation. These errors are almost always design errors (frequently requiring changes in multiple components) and typically are not repeatable even when the software is stimulated in the same apparent way. To provide adequate test coverage and resolve the statistically significant Heisen-bugs, extensive statistical testing under realistic and randomized usage scenarios is necessary.

Conventional software programs executing a single program on a single processor typically contained only Bohr-bugs. Modern, distributed systems with numerous interoperating components executing across a network of processors are vulnerable to Heisen-bugs, which are far more complicated to detect, analyze, and resolve. The best way to mature a software product is to establish an initial test infrastructure that allows execution of randomized usages scenarios early in the life cycle and continuously evolves the breadth and depth of usages scenarios to optimize coverage across the reliability-critical components.

The basic characteristics of a good metric are as follows:

1. It is considered meaningful by the customer, manager, and performer.
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. Ambiguity is minimized through well-understood units of measurement (such as staff-month, SLOC,)
4. It displays trends. This is an important characteristic. Understanding the change in a metrics value with respects 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.
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 of the data they process.

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

CRITICISMS

Software metrics tend to be used as an aid in judging the quality of software development. Metrics are relatively easy to produce, but their use as a management instrument has drawbacks:

- **Unethical:** It is said to be unethical to reduce a person's performance to a small number of numerical variables and then judge him/her by that measure. A supervisor may assign the most talented programmer to the hardest tasks on a project, which means it may take the longest time to develop the task and may generate the most defects due to the difficulty of the task. Uninformed managers overseeing the project might then judge the programmer as performing poorly without consulting the supervisor who has the full picture.
- **Demeaning:** "Management by numbers" without regard to the quality of experience of the employees, instead of "managing people."
- **Gaming:** The measurement process is biased because of employees seeking to maximize management's perception of their performances. For example, if lines of code are used to judge performance, then employees will write as many separate lines of code as possible, and if they find a way to shorten their code, they may not use it.
- **Inaccurate:** No known metrics are both meaningful and accurate. Lines of code measure exactly what is typed, but not the difficulty of the problem. Function points were developed to better measure the complexity of the code or specification, but they require personal judgment to use well. Different estimators will produce different results. This makes function points hard to use fairly and unlikely to be used well by everyone.
- **Uneconomical/Suboptimal:** It has been argued that when the economic value of measurements are computed using proven methods from decision

theory, measuring software developer performance turns out to be a much lower priority than measuring uncertain benefits and risks.

REFERENCES

[1] Stephen P. Berczuk (with Brad Appleton), 2003. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*. Addison-Wesley, 2003.

[2] John D. McGregor. The Evolution of Product Line Assets, Software Engineering Institute, CMU/SEI-2003-TR-005, 2003.

[3] Jitender Kumar Chhabra, K. K. Aggarwal and Yogesh Singh, Measurement of Object-Oriented Software Spatial Complexity. Information and Software Technology, **46** (10), (2004) 689-699.

[4] S. R. Schach, *Introduction to Object-Oriented Analysis and Design.* Tata, McGraw-Hill, 2004.

[5] Object Management Group. Software Process Engineering Meta-Model v. 1.1, 2005.

[6] K. K. Aggarwal, Yogesh Singh, Pravin Chandra, Manimala Puri: An Expert Committee Model to Estimate Lines of Code. ACM SIGSOFT Software Engineering Notes **30**(5): 1-4 (2005).

[7] www.eclipse.org. Eclipse Process Framework 1.0, 2006.

[8] Kannan Mohan and Balasubramaniam Ramesh. Change Management Patterns in Software Product Lines, Communications of the ACM, **49** (12), 2006.

[9] Software Engineering Institute, "Framework for Product Line Practice," http://www.sei.cmu.edu/productlines, 2006.

[10] Shakti Kumar, K. K. Aggarwal, Jagatpreet Singh: A Matlab Implementation of Swarm Intelligence based Methodology for Identification of Optimized Fuzzy Models. Swarm Intelligent Systems 2006: 175-184.

[11] K. K. Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra: Software Design Metrics for Object-Oriented Software. *Journal of Object Technology*, **6**(1) (2007).