MANNING

SAMPLE CHAPTER

Software Development Metrics

David Nicolette

FOREWORD BY George Dinwiddie

Software Development Metrics

by David Nicolette

Chapter 1

contents

foreword xi
preface xiii
acknowledgments xvi
about this book xvii

1 *Making metrics useful* 1

1.1 Measurements and metrics 2

What makes a metric "pragmatic"? 3 ▪ Forward-facing and backward-facing metrics 4

1.2 Factors affecting the choice of metrics 5

Process model 5 ▪ Delivery mode 7

1.3 How the metrics are presented 7

1.4 Name of the metric 7

1.5 Summary 8

2 *Metrics for steering* 9

2.1 Metric: Percentage of scope complete 11

When to use percentage of scope complete 12 ▪ A traditional project 12 ▪ An adaptive project 15 ▪ How to use percentage of scope complete 17 ▪ Anti-patterns 20

2.2 Metric: Earned value 21

When to use earned value 21 ▪ A traditional project 21 Anti-pattern: the novice team 24

2.3 Metric: Budget burn 24

When to use budget burn 24 ▪ A traditional project 25 An adaptive project using beyond budgeting 26 ▪ Anti-pattern: agile blindness 30

2.4 Metric: Buffer burn rate 30

When to use buffer burn rate 31 ▪ How to use buffer burn rate 31

2.5 Metric: Running tested features 32

When to use running tested features 32 ▪ An adaptive project 33 ▪ Anti-pattern: the easy rider 34

2.6 Metric: Earned business value 34

When to use earned business value 35 ▪ An adaptive project 35 ▪ Anti-patterns 38

2.7 Metric: Velocity 39

When to use velocity 39 ▪ An adaptive project 40 Anti-patterns 42

2.8 Metric: Cycle time 47

When to use cycle time 47 ▪ An adaptive project with consistently sized work items 47 ▪ An adaptive project with variable-sized work items 49 ▪ A traditional project with phase gates 50

2.9 Metric: Burn chart 52

When to use burn charts 53 ▪ How to use burn charts 53 Anti-patterns 55

2.10 Metric: Throughput 56

When to use throughput 56 ▪ A mixed-model project 57

2.11 Metric: Cumulative flow 59

When to use cumulative flow 59 ▪ A traditional project 60

2.12 Not advised 62

Earned schedule 62 ▪ Takt time 63

2.13 Summary 63

3 *Metrics for improvement 65*

3.1 Process-agnostic metrics 65

3.2 Technical metrics 66

3.3 Human metrics 66
3.4 General anti-patterns 66
Treating humans as resources 66 ▪ Measuring practices instead of results 67
3.5 Metric: Velocity 68
When to use velocity 68 ▪ An adaptive project 69 Anti-patterns 70
3.6 Metric: Cycle time 71
When to use cycle time 71 ▪ Tracking improvement in predictability 72 ▪ Tracking improvement in flow 73 Tracking responsiveness to special-cause variation 74
3.7 Metric: Burn chart 75
When to use burn charts 76 ▪ Adaptive development project using a time-boxed iterative process model 76
3.8 Metric: Cumulative flow 78
When to use a cumulative flow diagram 79 ▪ An adaptive project 79
3.9 Metric: Process cycle efficiency 82
When to use process cycle efficiency 83 ▪ Non-value-add time in queues 84 ▪ Non-value-add time in active states 85 What is normal PCE? 86 ▪ Moving the needle 86
3.10 Metric: Version control history 88
When to use version control history 88
3.11 Metric: Static code-analysis metrics 89
When to use static code-analysis metrics 89
3.12 Metric: Niko Niko calendar 92
When to use the Niko Niko calendar 92 ▪ Examples 92 Happy Camper 94 ▪ Omega Wolf 94 ▪ Zombie Team 95
3.13 Metric: Emotional seismogram 96
When to use the emotional seismogram 96 ▪ Examples 97
3.14 Metric: Happiness index 97
When to use the happiness index 98 ▪ Mechanics 98
3.15 Metric: Balls in bowls 102
When to use the balls-in-bowls metric 102 ▪ Mechanics 102
3.16 Metric: Health and happiness 102
When to use the health-and-happiness metric 103 Mechanics 103

3.17 Metric: Personality type profiles 105

When to use personality profiles 106 ▪ Anti-patterns 106

3.18 Summary 107

4 Putting the metrics to work 108

4.1 Pattern 1: Periodic refactoring iterations 109

4.2 Pattern 2: Velocity looks good, but little is delivered 113

4.3 Pattern 3: Linear workflow packaged in time-boxed iterations 121

4.4 Pattern 4: Erratic velocity but stable delivery 124

4.5 Summary 129

5 Planning predictability 130

5.1 Predictability and stakeholder satisfaction 131

Planning and traditional methods 131 ▪ Planning and adaptive methods 132

5.2 Measuring predictability 132

Estimation 132 ▪ Forecasting 134 ▪ Predictability of traditional plans 135 ▪ Predictability of adaptive plans 136

5.3 Predictability in unpredictable workflows 143

5.4 Effects of high variation in work item sizes 144

Deployable units of work 144 ▪ Trackable units of work 145 Demonstrating the value of consistently sized work items 145

5.5 Effects of high work-in-process levels 147

Work in process, cycle time, process cycle efficiency, and throughput 147 ▪ Work in process and defect density 152

5.6 Summary 154

6 Reporting outward and upward 155

6.1 Reporting hours 156

An example 157 ▪ Aggregate numbers are approximate 158

6.2 Reporting useless but mandated metrics 158

Categories of problematic metrics 159 ▪ Recognizing what's really happening 160 ▪ Beware of motivational side effects of metrics 161 ▪ Understanding what the numbers mean 162

6.3 Summary 163

index 165

1 Making metrics useful

This chapter covers

- The difference between measurements and metrics
- What we mean by *pragmatic* metrics
- Trailing and leading indicators
- The purpose and functions of metrics
- Factors to consider when choosing metrics

This book is designed for a person at the bottom of the management hierarchy in a software development organization. A person in such a position usually has direct responsibility for delivery as well as management duties at the team level. In a traditional organization, this role is usually called Project Manager. In contemporary organizations, people with similar responsibilities may have a title like Team Lead, Development Lead, Delivery Lead, Scrum Master, or Iteration Manager. In a peer-based, self-organizing team, these responsibilities may be shared across all team members.

The purpose of the book is to provide practical guidance to people who need to steer work in progress and who want to measure the effectiveness of process-improvement efforts. It offers a way to do so that doesn't depend on popular buzzwords and doesn't require the work to be done in any particular way. It suggests

what can be measured based on organizational realities, and not necessarily what should be measured in an ideal world.

Anything you do in the course of your work ought to have a clear purpose. Otherwise, you're just performing random activities in order to stay busy. Metrics for software development have a couple of purposes. First, you can use them to judge how well you're tracking toward the goals of a project. Second, you can use metrics to help you understand whether you're improving your delivery performance.

With that in mind, metrics can help with the following:

- Steering work in progress
- Guiding process improvements

Software development and delivery is usually carried out either as a discrete project that has a beginning and an end, or as an ongoing activity for evolutionary development or production support. In either case, there are expectations about how the work will progress. You need to know, as early as possible, when actual performance is diverging from expected performance so that you can take appropriate corrective action. I think of this action as steering the work: directing the work toward a goal.

It has become the norm for software professionals to assess their own practices and methods almost continuously, and to try to improve the way they do their work. Metrics can be useful to help you understand when a change leads to improvement and when it doesn't. Metrics can also help you make a case to change formal methods based on quantitative results from using a proposed new approach.

This chapter sets the stage for our examination of metrics for software development. To choose metrics appropriate to your work context, you need to know what decisions you're trying to support through metrics. You also need to understand how each metric is affected by a few key factors, such as whether you're taking a traditional or adaptive approach to development, what sort of process model you're using, and whether you're running discrete projects or carrying out continuous development and support.

1.1 Measurements and metrics

A *measurement* is a quantitative observation of one of the following:

- Something relevant to the decisions you have to make
- Information you have to report regarding the progress of development
- The effects of process improvements

A *metric* is a recurring measurement that has informational, diagnostic, motivational, or predictive power of some kind. It helps you understand whether you're at risk of missing expected results, or whether changes in process or practices are resulting in improved performance.

1.1.1 What makes a metric "pragmatic"?

Sometimes, managers get a bit carried away with metrics. They track all the metrics they can think of, or all the metrics their favorite project-management tool happens to support. They may or may not be able to tell you just why they're tracking any given metric. That sort of thing isn't practical; it's busywork. It's better to be pragmatic about measurement—that is, to have a clear purpose in mind for each metric you use.

There is effort and cost involved in collecting data and tracking metrics. To justify this cost, any metrics you use must have a practical purpose. A metric is *pragmatic* if it provides information that helps a stakeholder make a decision.

People usually think of the customer of a software product as the primary or only stakeholder of the software development project. For the purposes of this book, *you* are the main stakeholder, because you're the party with primary responsibility for tracking progress. Your management, other departments in your company, and members of the development team are also stakeholders.

Ideally, any metrics you track will help at least one of these stakeholders make decisions of one kind or another. Customers may make decisions about scope and schedule depending on how the work is progressing. Management may make decisions about portfolio management and budget allocations. Team members may make decisions about how to improve their delivery effectiveness. You may make decisions about how to steer work in progress.

All too often, project managers track metrics just because they can, or just because "it's always been done that way." I've seen managers get carried away with graphics or query options offered by their project-management software. I've seen others track metrics that don't apply to the work they're managing, because they used the same metrics on previous projects where those metrics *did* apply. And I've seen managers use metrics that formally belong to the methodology they *think* they're using, when in fact the work isn't done according to that methodology. I want to encourage you to consider the practical purpose of any metrics you choose to use, and to avoid creating extra work for yourself by collecting data that won't or can't be used to support decisions.

Trailing and leading indicators

We're interested in measuring things that have already happened as well as predicting things that are likely to happen in the future. Measurements of things that have already happened can often help us predict how things are likely to progress going forward.

Any metric that provides information about things that have already happened is considered a *trailing indicator* or *lagging indicator*. Any metric that helps us predict how things will happen in the future is considered a *leading indicator*. A leading indicator often comprises a series of trailing indicators along with a calculated trend that suggests how things are likely to play out, provided circumstances remain stable.

FUNCTIONS OF METRICS

Metrics have three functions, or effects:

- Informational
- Diagnostic
- Motivational

When a metric provides plain information, it serves an informational function. When a metric calls attention to a problem, it serves a diagnostic function. When a metric influences people's behavior, it serves a motivational function. Metrics may perform in more than one of these ways at the same time and can have effects that you didn't intend or plan—especially motivational effects.

1.1.2 Forward-facing and backward-facing metrics

There are a couple of different general approaches to software development and delivery. The *traditional approach* involves a thorough analysis of stakeholder needs, a comprehensive solution design, a careful assessment of risks, and a fixed budget allocation in advance. The *adaptive approach* involves defining a vision for the desired future state, performing sufficient analysis to get started, and exploring the solution space in collaboration with stakeholders through incremental delivery and frequent feedback.

Many metrics boil down to a comparison between expected and actual performance. With the traditional approach, the definition of expectations is in the comprehensive project plan that's created before development begins. As development progresses, the definition of success (the project plan) lies in the past. Even when a plan is re-baselined, the new plan lies in the past, from the perspective of the development team. I think of metrics that support traditional development as *backward-facing* metrics, because you have to face the past in order to see your target (see figure 1.1).

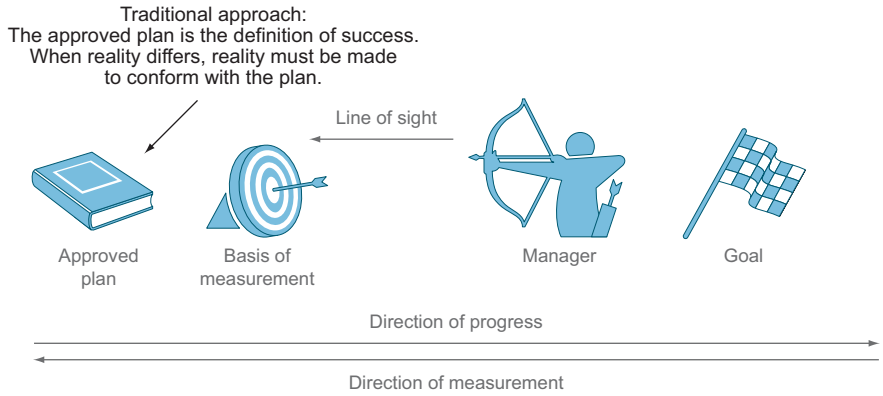

Figure 1.1 Traditional development: you must face the past to see your target.

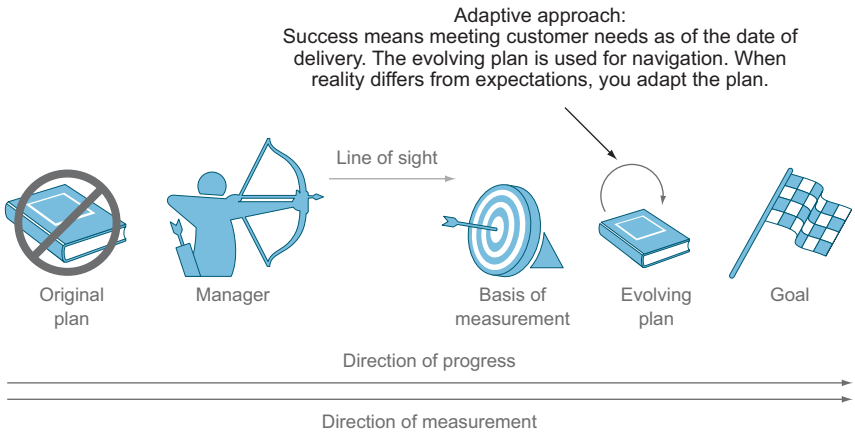

Figure 1.2 Adaptive development: you must face the future to see your target.

With the adaptive approach, the definition of expectations is the point-in-time understanding of the future-state vision as of today. This understanding evolves day by day as development progresses. I think of metrics that support adaptive development as *forward-facing* metrics, because you have to face the future in order to see your target (see figure 1.2).

For the purpose of choosing meaningful metrics, the key distinction is the way the *triple constraint* or *iron triangle* of scope, schedule, and budget is managed. With the traditional approach, the scope, schedule, and budget are all fully defined in advance. Metrics are used to track the development team's performance compared with the plan.

With the adaptive approach, one or two of these factors are left flexible on purpose. Metrics are used to assess whether the scope, schedule, or budget has to be adjusted to keep the work on track toward the future-state vision. Some metrics are meaningful only with one approach or the other.

1.2 Factors affecting the choice of metrics

In addition to the general approach—traditional or adaptive development—you also have to consider the *process model* and *delivery mode* you're using to develop and deliver the solution.

1.2.1 Process model

The sort of development process you're using will influence your choice of metrics. Some metrics depend on the work being done in a certain way. A common problem is that people believe they're using a given process, when in fact they're working according to a conflicting set of assumptions. If you apply metrics that depend on the process being done correctly, you won't obtain information that can help you steer the

work or measure the results of process-improvement efforts. You have to measure what's *really* happening, regardless of the buzzwords people use to label it.

Countless published and home-grown processes are in use to build and deliver software. In my experience, they all boil down to just four basic reference models:

- *Linear*—Based on the assumption that software development must proceed in order through a distinct series of steps. The steps include activities such as business analysis, requirements specification, solution design, coding, testing, deployment, and support. The linear process model is sometimes called a *waterfall* process, because work can't flow backward any more than water can flow uphill.
- *Iterative*—Based on the assumption that a single pass through the requirements is unlikely to result in a good solution. With an iterative process, the requirements are revisited time and again, and the solution is built up through a series of iterations. This may involve progressive refinement of the solution, gradual addition of specific features, or a combination.
- *Time-boxed*—The same as the iterative model, with the addition of two defining characteristics: (1) each iteration is the same length, and (2) a potentially shippable increment (or *vertical slice*) of the solution is delivered by the end of each time-boxed iteration.
- *Continuous flow*—Based on the assumption that the most effective way to keep work moving forward is to focus on maintaining a continuous flow, usually by controlling the level of work in process (WIP) and using techniques adapted from the Lean school of thought.

All real-world processes are based primarily on one of these four reference models and include elements from one or more of the remaining three. You can usually use metrics that apply to the reference model that is closest to the actual process you're using. As this is written, the iterative model is the most widely used and has the largest range of variations in practice.

If your organization is typical, then a couple of things are probably true:

- More than one process model is in use.
- Each process model in your organization is a hybrid model.

Software organizations of any appreciable size almost always apply different processes to different types of work, depending on the nature of the work. For example, you might use a linear process for highly predictable, routine projects; an iterative or time-boxed process for work that has to do with creating or maintaining competitive advantage; and a continuous-flow process for production support and infrastructure support.

In addition, only a vanishingly small number of organizations use any given process exactly as it's defined in books. Processes are almost always customized to the needs of the particular organization. Sometimes the modifications are well-reasoned

adjustments that take into account the local realities of the organization. Other times they're the result of misunderstanding how a process is meant to work, particularly when it's a relatively new process that's just becoming popular.

This book takes no sides on those issues. As a practical matter, the important thing for you is to be able to recognize how your work really flows and which metrics might help you steer.

1.2.2 Delivery mode

Software is built, delivered, and supported in one of two ways: as discrete projects or as ongoing development and support. A project has a start date and an end date. Between those dates, a team strives to achieve one or more goals—delivering a set of application features or standing up an IT asset. Some organizations form a new team for each project, whereas others assign projects to stable teams. Projects are often treated as capital investments and budgeted accordingly.

In an ongoing development and support mode, application or infrastructure features are delivered incrementally on an ongoing basis. Applications or technical assets are usually supported by the same team that enhances them. Ongoing work is often treated as an operating expense and budgeted accordingly.

In a corporate IT department, production support and operations are usually managed as ongoing support, whereas application development and new infrastructure features are usually managed as discrete projects. But ongoing delivery is also feasible for application development. Many internet-based email services, online catalog sales systems, social media sites, and other types of applications are developed and supported in an ongoing mode that has no planned ending date, sometimes called *continuous beta*. Some companies are finding this mode works well for all kinds of IT work and are moving away from discrete projects altogether.

Some metrics are sensitive to this factor and are meaningful with only one of these two options. The largest challenge when choosing metrics for steering work is the case when the same team has ongoing support responsibilities combined with project work—not unusual for infrastructure teams.

1.3 How the metrics are presented

Chapters 2 and 3 deal with individual metrics in isolation. We'll cover the purpose, mechanics, enabling factors, and common anti-patterns (inappropriate uses) of each metric. This is the format I'll use to describe each metric:

Name of the metric

Question(s) answered

- What does this metric tell us? It tells us *this* and *that*.

Description

- A brief description of the metric Value
- The value we can obtain by using the metric

Dependencies

- Approach: traditional or adaptive
- Process model: linear, iterative, time-boxed, continuous flow, or any
- Delivery mode: discrete project or continuous development

Success factors

- Special considerations above and beyond the basic dependencies

1.4 Summary

In this chapter, you learned the following:

- A *measurement* is a point-in-time observation of a single data point, whereas a *metric* comprises recurring measurements organized in a way that's designed to provide information useful for making decisions about your work.
- You use metrics for two purposes: to help steer work in progress and to help monitor the effectiveness of process-improvement efforts.
- Metrics have three *functions* or *effects*: informational, diagnostic, and motivational. Any metric can perform more than one of these functions simultaneously. Metrics often have a motivational effect even when you don't intend it.
- Your choice of metrics depends on three general factors: the *approach* (traditional or adaptive), the *process model* (linear, iterative, time-boxed, or continuous flow), and the *delivery mode* (discrete project or continuous evolution/support).
- The definition of success in traditional software development is to conform closely to a project plan developed in the past, sticking to the originally defined scope, schedule, and budget. Because the target lies in the past, to track progress you must use *backward-facing* metrics.
- The definition of success in adaptive software development is to deliver the business value that stakeholders require at the time they need it, at the appropriate level of quality, and at the right price point. Because the target lies in the future, to track progress you must use *forward-facing* metrics.

SOFTWARE ENGINEERING

Software Development Metrics

David Nicolette

When driving a car, you are less likely to speed, run out of gas, or suffer engine failure because of the measurements the car reports to you about its condition. Development teams, too, are less likely to fail if they are measuring the parameters that matter to the success of their projects. This book shows you how.

Software Development Metrics teaches you how to gather, analyze, and effectively use the metrics that define your organizational structure, process models, and development methods. The insights and examples in this book are based entirely on field experience. You'll learn practical techniques like building tools to track key metrics and developing data-based early warning systems. Along the way, you'll learn which metrics align with different development practices, including traditional and adaptive methods.

What's Inside

- Identify the most valuable metrics for your team and process
- Differentiate "improvement" from "change"
- Learn to interpret and apply the data you gather
- Common pitfalls and anti-patterns

No formal experience with developing or applying metrics is assumed.

Dave Nicolette is an organizational transformation consultant, team coach, and trainer. Dave is active in the agile and lean software communities.

"A real boon to those making the transition from a traditional serial development model to an agile one."
—From the Foreword by George Dinwiddie

"Provides a solid foundation for how to start measuring your development teams."
—Christopher W. H. Davis
Nike, Inc.

"Measurement is the key to making and consistently hitting scheduling targets. This book will help you confidently build a schedule that is accurate and defensible."
—Shaun Lippy
Oracle Corporation

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/software-development-metrics

ISBN 13: 978-1-617291-35-7
ISBN 10: 1-617291-35-8
54799
9 781617 291357

MANNING $47.99 / Can $55.99 [INCLUDING eBOOK]