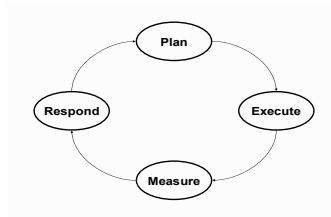


# Chapter 2

## Automated Metric-Driven Processes



### Introduction

Historically, many verification projects have been performed as an afterthought. They have been understaffed, under-planned, and under-executed. With today's complex design, it is widely agreed that verification consumes up to 70% of the total effort for a typical design project.

There are several issues that plague verification efforts. Among them are:

- *Insufficient planning.* High-priority issues are brought to light in the last stages of the project causing huge upheavals in resources and scheduling.
- *Lack of visibility.* Projects are frequently tracked by human updates. This is also known as “death by status meeting.”
- *Scheduling issues.* Why is it a well-accepted axiom that the last 20% of the work will consume 80% of the available time? Shouldn't the last 20% of the effort take 20% of the time?
- *Inefficient use of tools.* The EDA industry has promoted verification solutions for years. It's well accepted that the verification effort required 70% of the total design cycle effort. Why hasn't this number changed in years in face of the advanced solutions available?

In this chapter, we'll outline the basis for a methodology that will resolve these issues. This methodology is based on automated metric-driven processes. The methodology is enabled by a new class of tools called metric-driven process automation tools, or MPA tools.

Processes are an important start to our solution. By using repeatable processes we can improve the predictability of our projects. A framework for modeling processes will be described.

But, processes aren't enough. Without visibility into the workings of these processes we are unable to track progress and respond to issues in an efficient manner. We'll describe how to identify metrics that should be tracked during the life of the process. These metrics will give us constant insight into the process's progress.

Even well-defined processes and metrics that track their progress aren't enough. The nature of the metrics is also important. The classic 20/80 situation described above is an example of a metric-driven process that doesn't work. In this case the metric is an engineer's opinion of the completeness of a given task. The tracking mechanism is a query from management. In order to be useful metrics must be objective rather than subjective and be capable of being automatically captured and tracked. *We have to remove human interpretation and reporting of metrics from the equation.*

MPA tools facilitate the methodology described. They facilitate the planning phase by enabling users to define what metrics will be used while planning. They can control our verification engines removing that tedious time consuming task. They automatically capture the metrics that are produced by the verification engines. Finally, they offer analysis engines that can process the metric data. The analysis engines can be used in conjunction with the execution control aspects of MPA tools to completely automate some processes.

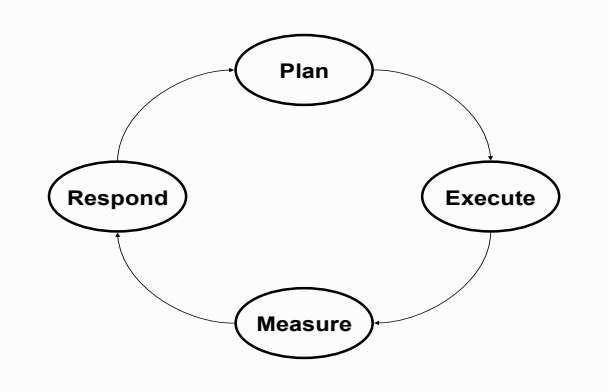
Using automated metric-driven processes, we'll be able to better plan our work defining exactly what needs to be done in a manner that's measurable. These automated measurements will allow us to efficiently respond to issues as they arise. We'll even be able to use

these metrics to further automate some of our manual processes and increase our operational efficiency.

Next we'll define the process model that will be used throughout the book.

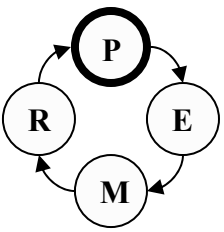
**The Process Model**

For the purposes of this book, a process is any activity that can be modeled using the flow shown in Figure 2.1.

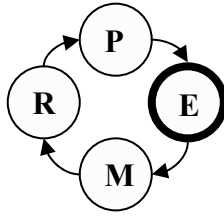


**Figure 2.1 Verification Process Model**

The flow consists of four phases. These are planning, execution, measurement, and response.

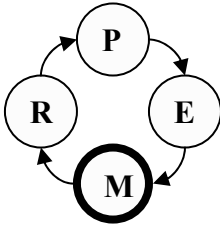


The first step of the process model is planning. This is where we determine what needs to be done and how to measure that it was in fact done. To efficiently execute a process, we need to know what the process hopes to achieve. Next, we need an objective way of knowing that the process has in fact achieved its goal.



Once the process is planned, we need to make it happen. That brings us to the execution phase of the model. During the execution phase, we will act on our plans. Using our available human resources and verification engines, we'll create the verification environments we specified during planning.

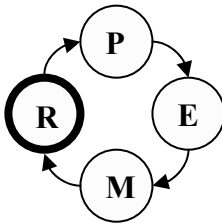
These environments will create objective metric output that we'll use to gauge the completeness of the plan. We will use MPA tools to control the execution engines.



As our engines operate we need to measure the effectiveness of our efforts. During the planning phase we specified the metrics that will be used to gauge this effectiveness. The MPA tool will automatically gather the specified metrics from our execution engines. Some typical metrics include:

- RTL code coverage
- Functional coverage
- Assertion coverage
- Software code coverage
- Error messages and types
- Revision control information

The user specifies how these metrics are to be annotated back to the plan during the planning phase.



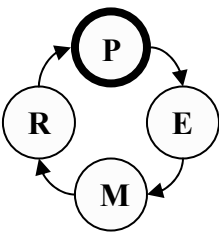
In the response phase, the user acts on the results of the data analysis performed during the measure phase. This analysis will be used to adjust existing plans and to facilitate or in some cases fully automate other verification processes.

For example, if bugs were found where none were expected using random testing, the user could respond by updating the verification plan to include explicit functional coverage that targets the areas where the bugs were found.

## The Automated Metric-Driven Process Model

Let's take a look at the process model in an automated metric-driven process. We'll discuss each phase individually outlining how each phase is related to the others and how each phase is enhanced by the new methodologies discussed in this book and by the application of MPA tools.

Planning



During the planning phase we will determine what needs to be verified and what metrics will be measured. There are many stakeholders in the verification process. Each stakeholder tends to have different concerns about the device. They each have their own perspective on verification. Some of these perspectives are shown in Figure 2.2.

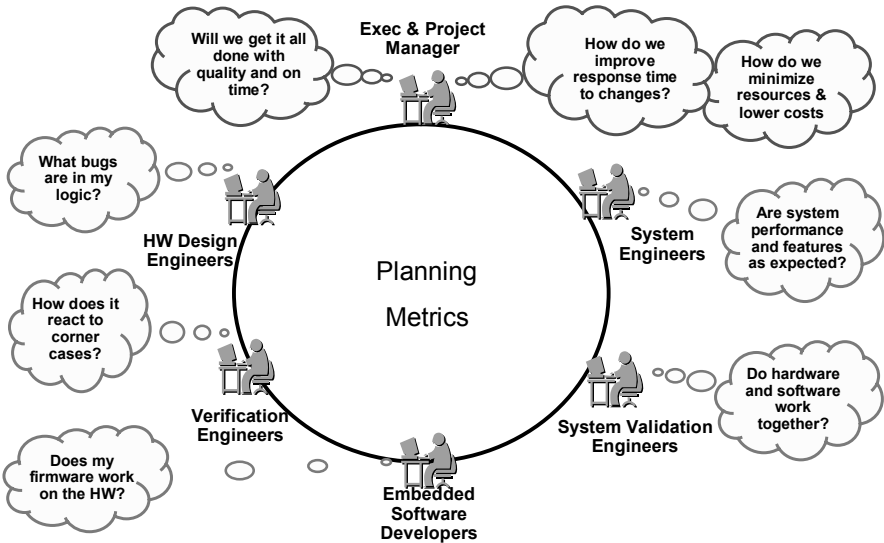


Figure 2.2 Stakeholder Perspectives

Verification concerns that are raised late in the project are one of the main causes of schedule slips. To avoid this, we involve *all* stakeholders in the planning phase. The planning technique used in the MPA methodology may be a bit different than what you are used to. It is a collaborative brainstorming effort. All stakeholders in the project participate in the planning session. During the session, the

device is discussed on a feature basis. The designer presents his section of the device based on what it does. Each participant is encouraged to contribute their concerns about a given feature to the discussion. Along with each concern, the participant works with the group to identify a metric that will guarantee the concern was addressed.

Each of these metrics must be *objectively and automatically measurable*. By using objective metrics, we remove human subjectivity from the equation. We know the exact status of our processes based on the metrics we have defined. By using only metrics that can be measured automatically, we ensure that we will always have real time status. Tracking process metrics is no longer an “extra” task that may get lost in the shifting priorities of a hurried project.

Let’s illustrate capturing concerns with a few examples. A design engineer is concerned that his DMA engine be exercised in such a manner that the input and output FIFOs are full simultaneously. An assertion that checks for this condition will provide a metric that addresses the concern (assertion coverage).

A verification engineer is concerned that every feature is exercised in every possible configuration mode. This concern can be addressed using functional coverage as a metric.

A firmware engineer is concerned that the DMA engine can move the appropriate OS code from the ROM to the instruction memory. This concern can be addressed using functional coverage as a metric as well. *Each of these metrics can be captured automatically from our verification engines.*

The output of the verification planning session is an executable verification plan. This plan will be used as the basis for determining what tasks should be executed as the project proceeds.

An example verification plan is shown on the next page. The concerns of the design, verification and software engineers are captured for a DMA engine in our device. The top half of the page shows the

plan as it is written during the verification planning interview. The bottom half of the page shows the plan as it appears after it has been read into an MPA tool and the coverage metrics have been collected from several runs of our verification engines (Figures 2.3 and 2.4).

## Features

### 1.1 DMA Engine

The DMA engine moves blocks of data between the various memories of the device and the external memory. The engine is configured via address mapped configuration registers.

#### Design

**cover:** /sys/rtlcodecover/dmamod/\*

#### Verification

**cover:** /sys/verif/dma/regreadwrite/\*

#### Software

**cover:** /sys/verif/dma/scenario/instructionmove/\*

Figure 2.3 Verification Plan Editing View

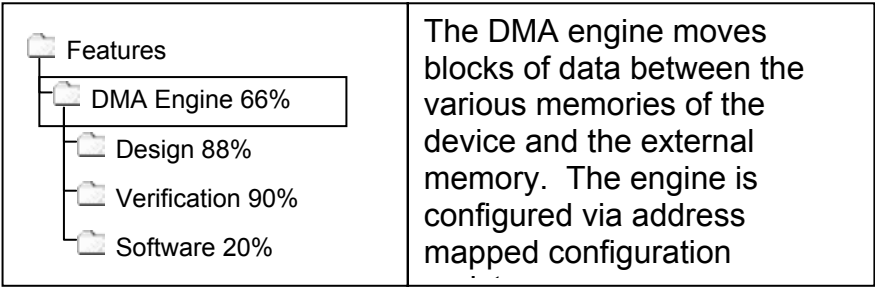


Figure 2.4 Verification Plan Executable View

As mentioned above, the planning sessions are collaborative. The value offered by the resulting plan is directly proportional to the number of stakeholders that actually attend the session. With the hectic pace of

chip design projects, it is often difficult to arrange for all stakeholders to attend these sessions. It should be stressed that these planning sessions are *extremely* valuable. They enable all the other techniques that will be described in the MPA methodology. Because of the collaborative nature of these sessions, several device bugs have been found while planning, without writing a single line of verification code.

### **Management Planning**

As the various stakeholders are outlining their technical concerns, management contributes by defining priorities for completion and schedule milestones. These priorities and milestones are captured in the plan as well. Priorities can be incorporated into the verification plan as weights on metrics corresponding to the key-prioritized features of the device. Most MPA tools allow milestones to be defined as well so that metric status data can be displayed along with defined milestones to make tracking the completion of the project more convenient.

For more information on defining weights and milestones see the chapter on planning in Part II of the book. It is very important to have appropriate reporting mechanisms organized and functioning *before* the project begins. Two of the key aspects of these reports should be the priority of different objectives and the milestones that are defined for their completion.

### **Visibility of the Plan**

One of the key requirements for metrics to actually be useful is visibility. That means visibility to everyone. In order for projects to come in on time, we need to make it impossible for anybody to “massage the status” either intentionally or not. Objective metrics go a long way in this direction.

*All* available metrics should be made visible to *all* the project’s stakeholders. By making these metrics available, we enable each contributor to creatively solve problems as they arise because they are aware of them. How many postmortems have you been to where an engineer said “Well, if we’d known what was happening, we could

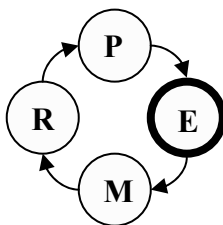


have executed the following process to solve the problem?” Wouldn’t it be nice if you never had to hear that statement again?

By automatically collecting objective metrics and allowing each of the users to personally interpret them we avoid two classic problems. First, automatically collected metrics do not create a resource drag on the project. No more walking from cubicle to cubicle to collect the daily status. No more interminable status meetings. We let computers do what computers are good at. As they run our simulation and emulation jobs, they automatically collect the metrics that we define as important. Second, we remove error-prone humans from the reporting process. The metrics collected are exactly and only the metrics created by our verification tools.

As discussed we’ll use automatically measured metrics to gauge the completeness of all our processes. In the execution phase, we’ll capture those metrics from our verification engines. For an in-depth explanation of the planning phase see Part II.

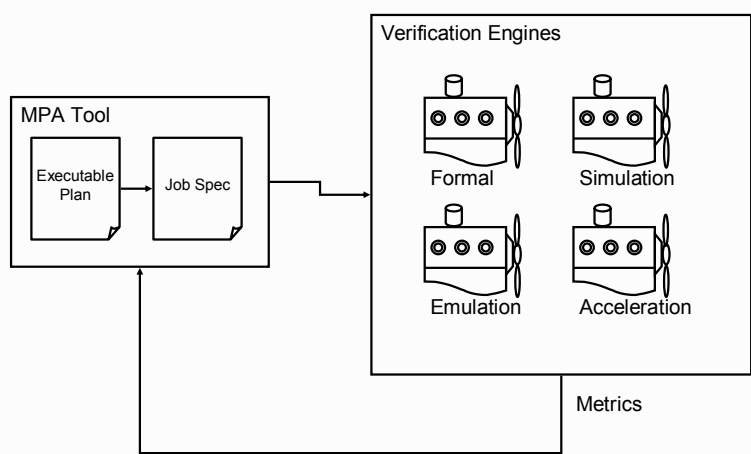
## Execution



During the planning session, we captured every concern and corresponding metric in an executable verification plan. In the execution phase, we’ll execute on those plans. There are two types of execution: implementation execution and verification engine execution. Implementation execution refers to the efforts made by engineers on the project team to implement environments that will run on the verification engines. Engine execution refers to the actual runs of the verification environments produced by the engineers.

Using a metric-driven approach allows us to improve the predictability, productivity, and quality of both implementation and engine execution. There are two opportunities that are presented by using metric-driven processes. First, using automatically capture metrics, we can get a better perspective on how our processes might be improved. Knowledge is power. Second, using automatically captured metrics, it is possible to fully automate some processes and remove the human element.

The execution phase at first glance seems simple, and it should. During this phase, we execute on our plans. During the execute and measure phases, our MPA tools will annotate the measured metrics specified in the verification plan to the metrics defined in the plan. Our verification plan will always have the latest status of all defined metrics embedded in the document. The flow for collecting metrics is shown in Figure 2.5.



**Figure 2.5 Automating the Execution Process**

Using the MPA tool, we start our verification engines. The MPA tool automatically tracks the metrics specified in the plan by extracting them from the outputs of the engines. These extracted metrics are then annotated back into the plan.

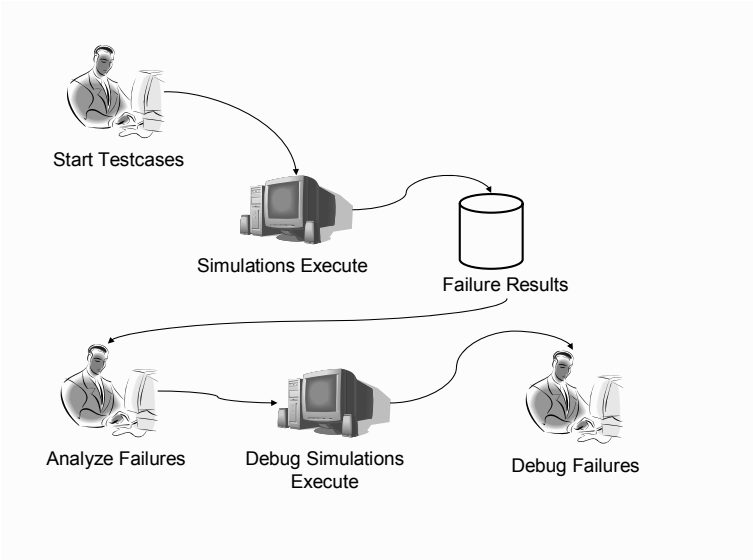
It is possible to detect problems earlier and better utilize resources because metrics can now be automatically captured and analyzed. With some advanced planning, teams can begin to solve a number of problems by collecting data that illuminates both the causes and solutions of those problems.

Using metric-driven concepts some time consuming, tedious processes can even be fully automated. Generation of detailed debug information is a good example of this.

Simulation time is valuable and should be used efficiently. After implementing or changing a design, the engineering team runs large sets of simulation testcases, (called regressions), to ensure the design has been implemented correctly. Once the current implementation is deemed acceptable by passing these testcases, more implementation can begin. Because of the iterative nature of this implementation process, it is desirable that these test suites execute as quickly as possible. To increase execution speed, most of the debug features of the simulator are turned off. Typically the only failure or debug information available in this mode is a brief message describing the failure and the time that it occurred.

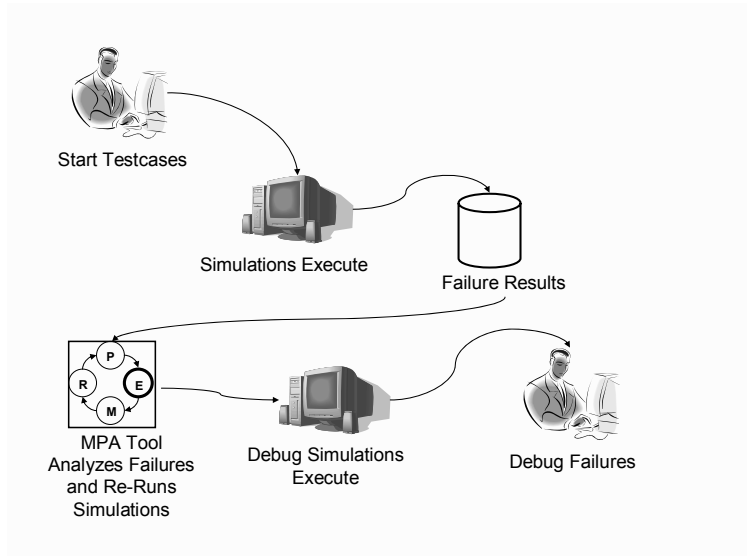
However, to completely debug an issue, an engineer needs much more information, such as waveforms that illustrate the signal levels of interest around the time the issue was detected. To gather this information, an engineer will manually sort through the failing testcases determining which testcases produce unique failures in the shortest amount of time and then run these simulations again with the waveform generation feature of the simulator turned on. This process is shown in Figure 2.6 with the human intervention points clearly marked.

By using our MPA tools to automatically analyze our captured metrics, we can completely automate this process. The technique is simple. The MPA tool captures the various unique failure types and then determines which testcases produced the failures in the shortest amount of time. Then, because the MPA tool has access to all the information required to start a given testcase, it can restart the pertinent simulations with debug features such as waveforms turned on. As a result, our engineers no longer spend hours analyzing failures and then waiting for the simulations to rerun.



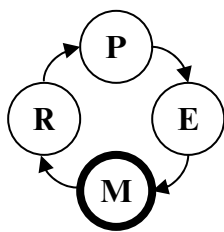
**Figure 2.6 Manual Debug Process**

They simply start the simulations once and then analyze the failure data as soon as the simulations complete as shown in Figure 2.7.



**Figure 2.7 Automated Debug Process**

Measurement



The measurement and analysis phase of the process is one of the most automated phases, and provides the bulk of the power offered by the MPA methodology. During the measurement phase, the MPA tool automatically captures and stores all the metrics that the project team has declared during the planning stage. MPA tools such as Cadence’s

Incisive Enterprise Manager ship with built-in metric-capture mechanisms for popular verification tools such as simulators and emulators. These capture modules automatically scan the output of simulator tools and extract common metrics such as failure messages, the amount of CPU time consumed by the simulator and the amount of real time consumed by simulator execution. By building easy to implement metric capture plug-ins, engineering teams can capture any other objective metrics from the outputs of their verification tools.

Using these metrics, MPA tools can automate standard analysis tasks as well. Using the example from the execution section earlier, from our simulation runs, the MPA tool captures:

- Failure type
- Failure time
- Testcase name

These metrics might be stored in a table (Table 2.1).

Table 2.1 Simulation Failure Metrics

Testcase Name	Failure Time	Failure Type
Dmaengine1	1000	FIFO pointer assertion
Dmaengine2	200	FIFO pointer assertion
Dmaengine3	1025	FIFO pointer assertion
Dmaengine4	257	Bad read/write pair

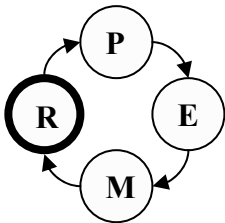
Using these metrics, the MPA tool can first group on the failure type and then sort on the failure time to determine the shortest set of testcases that can reproduce all the failures with debug information turned on. The results of this analysis are shown in Table 2.2.

Table 2.2 Failure Metric Analysis Results

Testcase Name	Failure Time	Failure Type
Dmaengine2	200	FIFO pointer assertion
Dmaengine4	257	Bad read/write pair

Engineers can define and store automated analysis tasks such as the one above. The MPA tool can then automatically perform these tasks where appropriate.

Response



During the response phase, our human resources re-enter the picture to do what they are best at: develop innovative solutions to improve the status of the project as revealed by our automated metrics capture and analysis.

Using our MPA tools for automated analysis, we can get project status such as that shown in Figure 2.8.

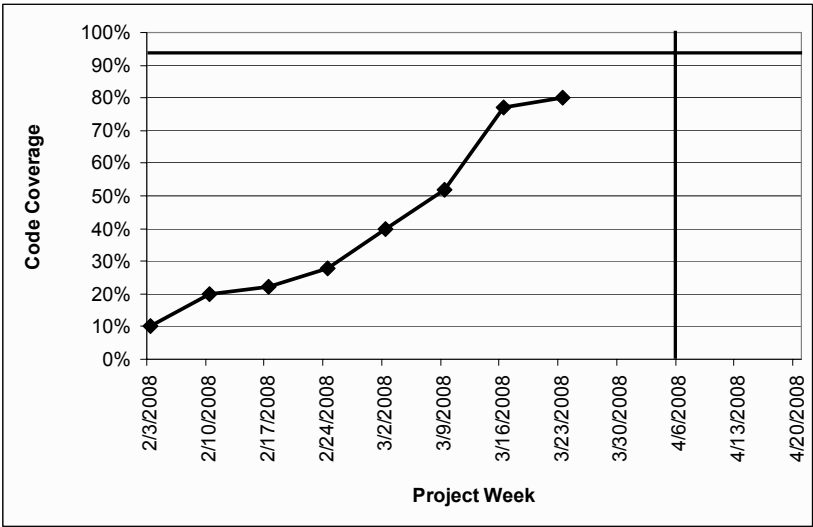


Figure 2.8 Project Wide Code Coverage vs. Project Week

With this information, our management team might judge that everything is on track. However, the data in Figure 2.9 tells a different story.

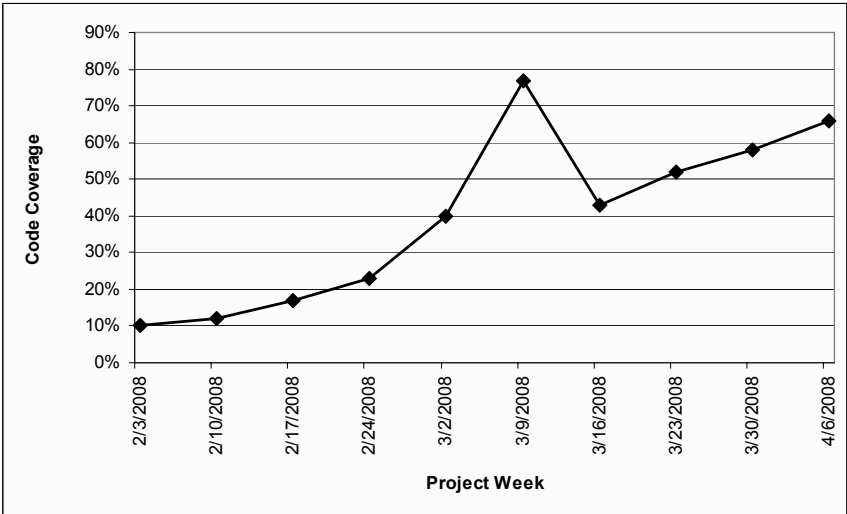


Figure 2.9 DMA Code Coverage vs. Project Week

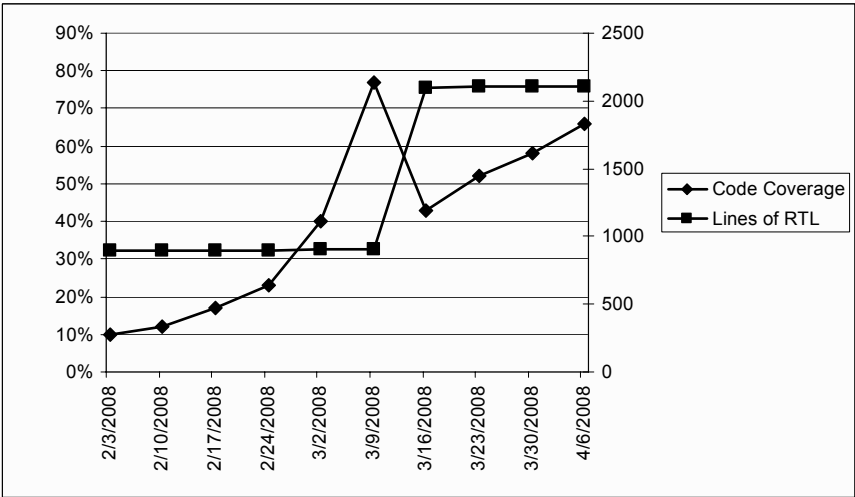


Figure 2.10 Code Coverage and Number of Lines of Code

We can see that there was a large reduction in code coverage for the DMA block on 3/16. By itself, this should be a red flag that indicates action needs to be taken with respect to the DMA block to avoid placing the schedule at risk. Using other automatically capture metrics, we can gain even more insight into the project status.

In Figure 2.10, we can see that the reduction in the total code coverage was caused by a large increase in the number of lines of code implemented. This may have been caused by a new feature being added to the design. In such cases, management can now make judgments regarding the cost of adding the feature based on real data rather than impressions or opinions. While decision should *always* be based on real data, it is much easier to follow this axiom when the real data is readily available.

## **Project Management Using Metric-Driven Data**

Hundreds of books have been written on project management. Everyone has their favorite methodology. Almost all these methodologies have one thing in common: the raw material for making decisions is objective data collected from the execution of the project itself. This data is exactly what metric-driven methodologies provide. Rather than trying to expound on our favorite project management methodology here, we'll respect your choices. This book tells readers how to determine what metrics to measure and how to objectively and automatically gather those metrics to enable your project management methodology of choice.

As mentioned earlier, the paramount task before the readers is to make sure that their metrics are properly utilized. As part of the planning process, the management team should identify all status reports that they will need to properly evaluate and manage the project. These reporting mechanisms should be setup and tested *before* the project begins. This small amount of effort early in the project will enable all the gains discussed throughout the remainder of the book.



## **What Are Metrics For?**

Knowledge is power! Our metrics will increase the power of the project team in several ways. First, metrics give the team the chance to react to dynamic requirements changes. As resources and requirements change, subsequent changes in measured metrics allow us to detect and react to these changes. Second, as illustrated above, metrics can be used to completely automate some processes.

## **Tactical and Strategic Metrics**

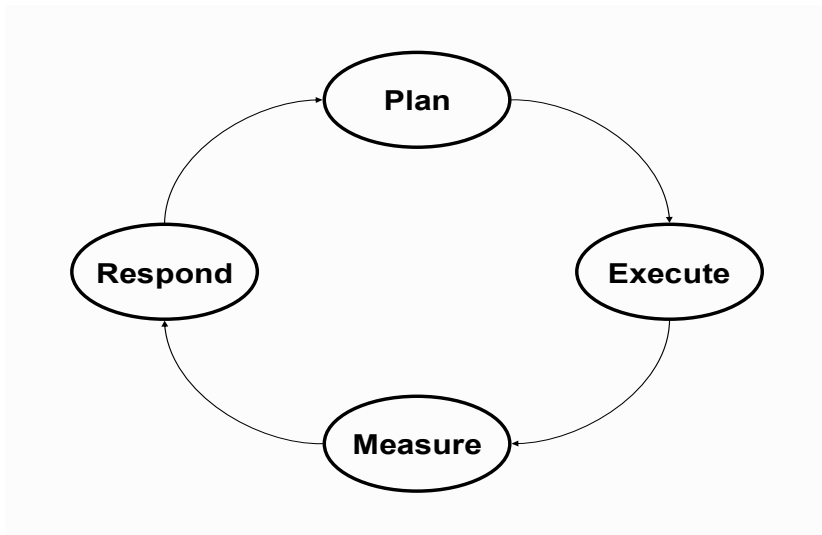
In this book, metrics will be placed in two broad categories. The first set of metrics is tactical metrics. These are metrics that give the team data about where the project stands at present. Tactical metrics, as their name implies, are used to make tactical decisions during the course of the project. Examples of tactical metrics are the number of testcases that failed in the most recent regression, the percentage of coverage completeness on the serial block, and the number of assertions that activated successfully for each module in the chip.

The second set of metrics is strategic metrics. These are frequently referred to as historical metrics. Historical metrics may start out as tactical metrics, but in this context, they are tracked throughout a project and then used to make strategic decisions at a later stage of the project, or in a follow-on project that reuses aspects of a previous project. Historical metrics include the number of issues found in a module during development, the rate at which coverage closure was reached on a project, and the frequency that a module was revised over the course of a project.

Historical metrics can also be used to shape training and career development plans for members of the engineering team. They can be used in much the same way that professional athletic coaches use game statistics to determine where to focus the next weeks' practice sessions. We'll talk more about this in the verification management chapters.

## Summary

In this chapter we have illustrated the evolution of and justification for a metric-driven verification process. This process can be divided into four steps as shown in Figure 2.11.



**Figure 2.11 The Verification Process**

We will use this diagram throughout the book to help illustrate what portion of the process is being described. We looked at some of the key points of each of these four phases, and showed how the MPA methodology offers improvements in each phase.

<http://www.springer.com/978-0-387-38151-0>

Metric Driven Design Verification

An Engineer's and Executive's Guide to First Pass  
Success

Carter, H.B.; Hemmady, S.G.

2007, XXVII, 361 p. 178 illus., Hardcover

ISBN: 978-0-387-38151-0