Viewpoint
Making the Case for a “Manufacturing Execution System” for Software Development

Today’s software development process is supported by many tools such as integrated development environments, configuration management systems, issue tracking databases, requirements management systems, estimation tools, time accounting applications, as well as enterprise resource planning systems for human resources and finance data. What is missing is information integration with the goal of enabling software development management to make informed decisions that cut across the stovepipes of different tools and stakeholder concerns. Without such information integration, today’s software managers are doubtlessly making suboptimal decisions about productivity, quality, and resource allocation.

Our vision is inspired by the model of a manufacturing execution system (MES) (see Figure 2), which is used in manufacturing industries. The success of MES systems relies upon integrating data from various aspects of the manufacturing process.

Based on the data continuously collected and integrated throughout the process, an MES can aid in planning and estimation accuracy, support of process improvement through met-
Figure 1. MES areas and data dependencies.

Figure 2. Information integration from multiple sources to determine total cost of ownership of a feature.

Figure 3. Information integration from multiple sources to track and trace defects.

Given recent advances in software management tools, we observe that most information needed for a software MES is actually becoming available. To illustrate how the benefits of such information integration could be realized in software development management, we describe five scenarios that could be enabled by such a “software MES.” Each of these scenarios ties together data from different tools, providing insight that is largely unachievable today.

Scenario 1: Total Cost of a Feature

The total cost for a feature consists of design, implementation, test, and deployment costs, as well as the costs to fix defects or other maintenance work over its lifetime. In addition, there might be costs related to marketing, standardization, certification, or recruiting and retention of certain specialists related to that feature.

To calculate the total cost of a feature, it is necessary to connect the feature’s requirements in the requirements management system with resulting work items. The work items need to be connected to entries in the developer accounting and billing system. We need to separately...
track activities like design, implementation, and test to different cost categories in the time accounting software. After initial development, defects from the issue tracking system need to be connected to features as well as activity-type effort and billing information (see Figure 2). To facilitate such a correlation, a software MES will use a common information and identifier model.

Knowledge about the total effort and resulting cost of a feature would be useful for product management, for setting development priorities, and for pricing the feature for customers as well as for development management to improve future effort estimates, for example, to map story points to effort and cost values. It will be especially interesting to compare parts of the total life cycle cost of a feature relative to other features.

**Scenario 2: Developer Assignment According to Expertise and Personal Growth Plans**

On-the-job training of developers as well as the formation of a cross-trained team requires that developers get opportunities to work on tasks where they can build new expertise. This is typically captured in annual goal-setting documents and related HR systems, but such targets are often overlooked when the teams for a development task are assigned, because the person doing the resource assignment may not be the developer's line manager and does not have visibility into the HR competence and objective management systems, and because the cost for such cross-training cannot be estimated due to the lack of readily accessible historical figures.

A better system would combine the project resource allocation and scheduling applications with the HR system so that data about staff capabilities, levels of expertise, and productivity are automatically considered. In addition, the assignment of developers without relevant expertise, if such involvement is planned for their personal growth, can be planned.

Prevailing software management processes are currently separated from HR information, and developers' expertise and productivities over time are not tracked in current software management tools.

**Scenario 3: Compliance Documentation**

Software-based systems today must comply with many regulations, and this aspect will become even more important in the future. Compliance relates, for example, to intellectual property ownership and licensing, security and safety properties, as well as other legal and contractual obligations, such as export and arms control. The required evidence for compliance can be connected to development staff, processes, and artifacts.

To provide compliance information, every change set in the configuration management system must be connected with provenance documentation as well as information about any safety, security, or other certifications that apply to this version of the artifact. In addition, every modification of an artifact must be associated with links to all persons involved in the creation, test, review, and release of the version. It must be possible to extract, from the HR...
system, information about those persons, including their training, certification, and licensing records, as well as attributes such as their nationality or level of security clearance.

It might not be known, in advance, what kinds of compliance documentation will be needed later in the product life cycle. Thus it is important to have a flexible query interface that supports the creation of compliance reports from all cross-linked data sources. Prevailing software management tools focus on the tracking of software artifacts, but comprehensive information about people and associated compliance items is largely missing.

**Scenario 4:**
**Traceability of a Defect**

In quality-critical manufacturing—for example, pharmaceuticals or food products—it is possible to trace each quality issue to determine which people, machines, ingredients, and processes were involved and who else could be affected by the same issue. This same capability would be desirable for critical software products (see Figure 3), which means for every artifact involved in a defect, it must be possible to determine all persons and tools (such as compilers) that were involved in the development. In addition, one should be able to identify all other development artifacts that were created by the same tools and/or persons to check whether similar defects exist there as well.

Furthermore, it would be desirable to have a link between the software configuration management system and the license management database recording each deployment of the software, so that all customers/users of a certain problematic version or configurations can be informed about the problem and possible fixes, if necessary.

**Scenario 5:**
**Managing Technical Debt**

For any long-living and complex system, it is important to keep track of the health of the underlying subsystems and components. This is true for manufacturing plants as well as for software systems. While software, unlike physical assets, does not objectively deteriorate over time, it “ages” through increasing divergence of expected and actual levels of quality. The phenomenon of software maintainability degrading over time, primarily due to short-term-focused bug-fixing and feature-adding activities, has been described as a type of “technical debt.”

“Shall we spend time refactoring or just keep adding new features?” is a recurring dilemma for software managers. Dealing with this problem strategically requires optimal investment decisions for use of limited software maintenance budgets. To support such decisions, we first need to identify and quantify the trouble spots—areas of technical debt—by calculating effort, process metrics, and structural complexity metrics from a project repository. For example, we could monitor per file the average time needed to fix a bug or make a change, how coupled it is to other files, and whether it is historically implicated in many high-severity bugs. Similar to the previous scenarios, we can extract such information from the production process of a software MES, concretely, the source code repository, version control, and issue tracking systems.

But just identifying potential trouble spots is not enough. To make optimal decisions quickly, management has to balance multiple factors, additionally taking into account the future quality of the product, impending requirements and associated deadlines, the available skills of developers, and the costs of development, maintenance, and operations, which involves multiple areas of a software MES. In the end, virtually every decision in a software project is an economic decision, and such decisions should be made based on a comprehensive software MES.

**Conclusion**

While the recording of the information underlying the scenarios described in this Viewpoint may not be standard practice in all software companies today, most of it is actually available. The challenge is to integrate information from heterogeneous data produced by incompatible tools, and realize traceability throughout the development process. Currently, correlating all the necessary information is expensive and awkward, but not impossible. We have already begun to realize one of these scenarios in a research decision-support system that aids in identifying and measuring modularity debt to support refactoring decisions. Microsoft launched a similar information integration project called CODEMINE.

Clearly, much more work is required to make this entire vision a reality. This is a call to action for the software community.

---

**References**


