

End-to-End Defect Modeling

Jean-Jacques Gras

People with a scientific background tend to analyze and drive a system on the basis of observations and measurements. In software, however, only half of this statement seems true. We're usually convinced that we need to measure the performance of our processes, but we rarely see the benefits of driving them by data, despite recommendations of renowned maturity models.



To use an analogy: When you're learning to drive a car, your mind collects data and builds a sort of cause-effect model that helps you anticipate actions and predict results. In that scenario, learning and driving are relatively easy because you experience the result of your actions immediately. Driving a software project is more like piloting a large cargo ship—you need more time (or distance) to see the results of your actions, the complexity is higher, and the sea is more unpredictable. So learning and driving become more difficult. In this context, computer models are a must—they can help you predict outcomes and anticipate with confidence. Thankfully, you can now use cause-effect modeling to drive software quality, moving your organization toward higher maturity levels.

Successful software projects

Despite missing good software quality models, many software projects successfully deliver software on time and with acceptable quality.

Although researchers have devoted much attention to analyzing software projects' failures, we also need to understand why some are successful—within budget, of high quality, and on time—despite numerous challenges. In these projects, managers are in control. They not only can detect deviations from target but also initiate effective corrective actions. Successful process control supposes a good understanding of the relationships between inputs, process drivers, and output.

We can successfully determine corrective actions only if we know the process response to stimulation or change in its inputs and drivers. This knowledge must then exist in projects that perform as planned. Restricting software quality to defects, decisions made in successful projects must be based on some understanding of cause-effect relationships that drive defects at each stage of the process. To manage software quality by data, we need a model describing which factors drive defect introduction and removal in the life cycle, and how they do it.

Once properly built and validated, a defect model enables successful anticipation. What it lacks in richness compared to the "mental" model we use for intuitive decision-making, it gains in repeatability and objectivity. We can analyze its predictive performance and compare it to other models. No model is perfect, but it should explain with a reasonable level of confidence a process's outcome given measures of inputs and key drivers. This is why it's important that the model include all variables influencing the process response to some degree.

Causal models

We find most organizations are primarily interested in measuring the process response directly linked to business performance. To build our models, we need to measure simultaneously the variables controlling the process, the key drivers, the model's inputs, and the defects discovered along the development process until well after delivery.

In existing metrics databases, we don't usually find data related to all these factors and can't simply analyze data to produce a valid model for each activity in an organization's development process. Unfortunately, software development's specificity and economics make it impractical to run measurement campaigns producing the necessary data in a reasonable amount of time.

What can we do, then, when we don't have enough data to characterize our processes? We can capture the knowledge we mentioned earlier: that of cause-effect relationships between inputs, drivers, and process response—the knowledge that enables successful projects to deliver software according to plan.

Bayesian Network (BN) models have been proposed to capture this sort of knowledge because they offer the capability to describe cause-effect relationships in an easily accessible graph and to rigorously quantify the relationships with conditional probabilities.^{1,2}

During the modeling process, we elicit all the key factors and their cause-effect relationships, build a graph, and quantify the relations from expert opinions or from data (when available in large enough quantities). In large organizations, the knowledge we want to capture is most often distributed among the people involved in software development: engineers, team leaders, project managers, testers, and process improvement teams.

The graph provides an ideal way to share captured knowledge among all people involved in model construction.

Using defect models in development and test

This modeling approach with BNs is



a promising and practical way (see the sidebar) of driving software quality across the software systems' life cycle. During project planning, we can load into the model preliminary information about development and test factors that is available at this early stage. Thanks to Bayesian statistics, this limited information is complemented by "prior" probabilities for the missing factors derived from past projects and preset in the models. The resulting defect predictions are precious for refining the project plans, using what-if scenarios to assess the impact of changes in controllable quality factors. Further into the development, actual data about the missing factors can be fed to the models that can then produce better predictions.

An important feature of BN models is their ability to propagate information both ways: causes to effects and effects to causes (inference). When outcomes of the first verification and validation activities become available, we can force actual numbers of defects found into corresponding model out-

puts. Back-propagation will then revise the model's internal variables (such as the total number of faults inserted in each software component's code), which are unobservable at this stage. Consequently, we get more accurate predictions of latent defects before they escape to system integration and test.

We can also combine preliminary defect predictions available during component development with models describing component interactions, derived from system design, to generate system integration and test plans. We get, at an early stage, a way to calculate the optimal number of test cases to be written in each test area for each system feature. Revised predictions obtained after release of software components to system test teams are accurate enough to select the most efficient test cases and drive testing toward the weak spots. After each test stage, we can apply the same updating concept through back-propagation of actual defects found. This work leads eventually to a thorough ranking of system features, predicting how likely they are to fail in customers' hands. Associated

End-to-End Defect Prediction Models at Work

Prompted by the work of Norman Fenton and Martin Neil,¹ our team at Motorola Labs collaborated with several software organizations in Motorola over the years and found it possible to reach consensus on key software quality drivers and their relationships with the defects escaping each development and test activity. We built Bayesian Network models (BNs) corresponding to the main software development activities in the telecom domain, from the requirement phase to system test. The key drivers identified so far relate to people, process, and product factors. A BN model doesn't need to be an exact representation of the process. Much more simply, the graph describes only the fundamental relations between key drivers and process output attributes. Interestingly, and this is probably due to the fundamental character of the factors and relations, we received similar views from various development groups. Factors and graph structures were so close that it encouraged us to create a library of model components by merging models for equivalent activities but from different origins into more generic ones.

Defect models drive verification and validation


The development of large systems is split into components assigned to different teams. Each team uses a model to predict latent defects released to system integration and test (Figure A). Individual BNs predict the number of defects inserted during creation activities and removed by each verification and validation activity. They're connected to each other through dynamic links by our tool, the Bayesian Test Assistant, to map the particular process of a given component project. BTA takes data collected during development to produce a prediction of the defects left at each stage of the process.

End-to-end models drive system testing

Latent defects released by each component team to system integration and test are compounded by system service models (derived from the system architecture) to generate test plans long before actual integration starts (Figure B). These plans suggest how many test cases you should write to catch defects potentially exposed by each service's execution. Later, when integration is done, you can further use BTA to prioritize test cases for optimal test execution. Eventually, predictions of field defects' impact provide good support for release decisions.

with failure impact, these predictions are most useful to support release decisions.

BNs provide a practical way of transitioning from a world driven by intuition toward one driven by data. We can now build models of defect introduction and removal across

the whole life cycle that work well enough to drive verification and validation activities from development to system testing. Future research could target the creation of a library of generic, reusable model components for composing defect models that can fit in multiple domains and adapt to changing contexts. 

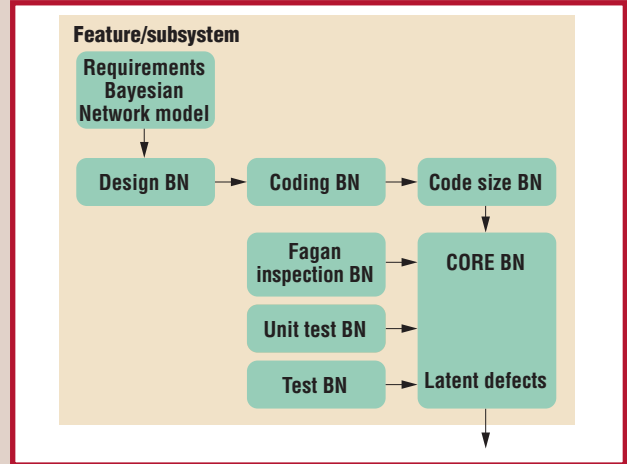


Figure A. A component defect model.

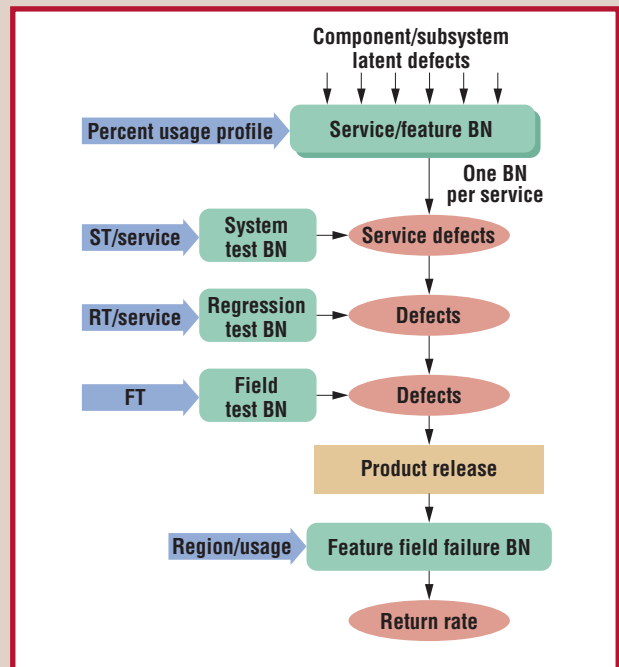


Figure B. An end-to-end model.

Reference

1. N. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," *IEEE Trans. Software Eng.*, vol. 25, no. 5, 1999, pp. 675-689.

References

1. N. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," *IEEE Trans. Software Eng.*, vol. 25, no. 5, 1999, pp. 675-689.
2. N.E. Fenton, P. Krause, and M. Neil, "Software Measurement: Uncertainty and Causal Modeling," *IEEE Software*, vol. 19, no. 4, 2002, pp. 116-122.

Jean-Jacques Gras is a principal research engineer at Motorola Labs. Contact him at jjgras@motorola.com.