Modeling Defect Trends for Iterative Development at JPL

John D. Powell

Caltech, Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena CA 91109-8099

The Jet Propulsion Laboratory's (JPL) Software Quality Improvement Measurement and Benchmarking (SQI-M&B) subgroup is charged with the responsibility of developing, infusing and supporting quantitative analysis of software metrics to foster improvement of existing and future software projects and the institution as a whole. A specific thrust within this effort is the measurement, analysis and improvement of defect trends during development of software at JPL. This paper will describe the approach being taken towards those ends.

The iterative software development lifecycle has largely replaced the waterfall lifecycle at JPL. Development of engineering models, including defect trend models, for this iterative environment requires a fundamental shift in the view of the role of defects and defect repairs within the development lifecycle. The treatment of defects within a waterfall lifecycle is solely as an unwanted byproduct of development. Conversely, iterative lifecycle's approach to learning about a software system while building it necessitates reliance, to some degree, on the discovery of defects to facilitate that learning. Thus, defects are a necessary development and management tool in the iterative lifecycle. This new reliance in an item that ultimately is bad for the software system, if allowed to go uncheck, demands rigorous management of defects to ensure the delivery of high quality software on time and within budget. JPL's SQI-M&B team has taken a quantitative analysis approach, based on specific software metrics, to aid JPL software projects in defect management. The approach calls for the "employment" of defects to aid software development as a form of in-process knowledge acquisition about the system in addition to their eventual removal. The models in the employment of defects approach seek to be a means to this end. Employing defects within an iterative style of software development seeks to formally:

- Make use of defects as a source of systematic learning throughout the lifecycle by formulating distinctions within the set of defects and their characteristics and then tracking and analyzing data regarding these characteristics over time from past JPL software projects through current and future projects.
- Capture the characteristics that account for the developmental flexibility, which may benefit or endanger (depending on their usage) software projects developed in JPL's iterative style. Then, form baselines to discriminate between the two possible effects probabilistically. These include but are not limited to proper deferment of defect repairs to later lifecycle stages in an attempt to maximize quality and minimize programmatic risks.
- Operate in conjunction with predictive defect models, such as COQUALMO, as a means of prediction across a within and across multiple projects over time.

This defect approach augments the waterfall style of predictive models like COQUALMO with defect measurement and analysis approaches that account for the shift in the development team's learning during development from a concentration in the requirements phase, before the software system development begins (strict waterfall), to a dispersion of learning throughout the phases and iterations of development (iterative). Currently, many JPL projects are forced to treat iterations in JPL's iterative lifecycle as a series of "mini-waterfalls" for defect analysis. This paper will present the reasons why the "mini-waterfall" mapping is an improper defect rationale and fails to capture the sufficient/correct information in an iterative environment to enable reliable predictors and goes on to show how the employment approach to defects overcomes these barriers. These measures and techniques will be specifically related to iterative development's intentional propensity to delay the full completion of artifacts until more is known about the system. Finally, it will show why defects should be viewed as a valuable source of developmental feedback even though they must be expected in significant levels as a result of developing software iteratively.