Software Reliability Modeling in the age of Continuous Integration / Continuous Delivery Paper #171

Tuesday, 24-Jan 8:00AM – 10:00AM

James T. Heires, PMP

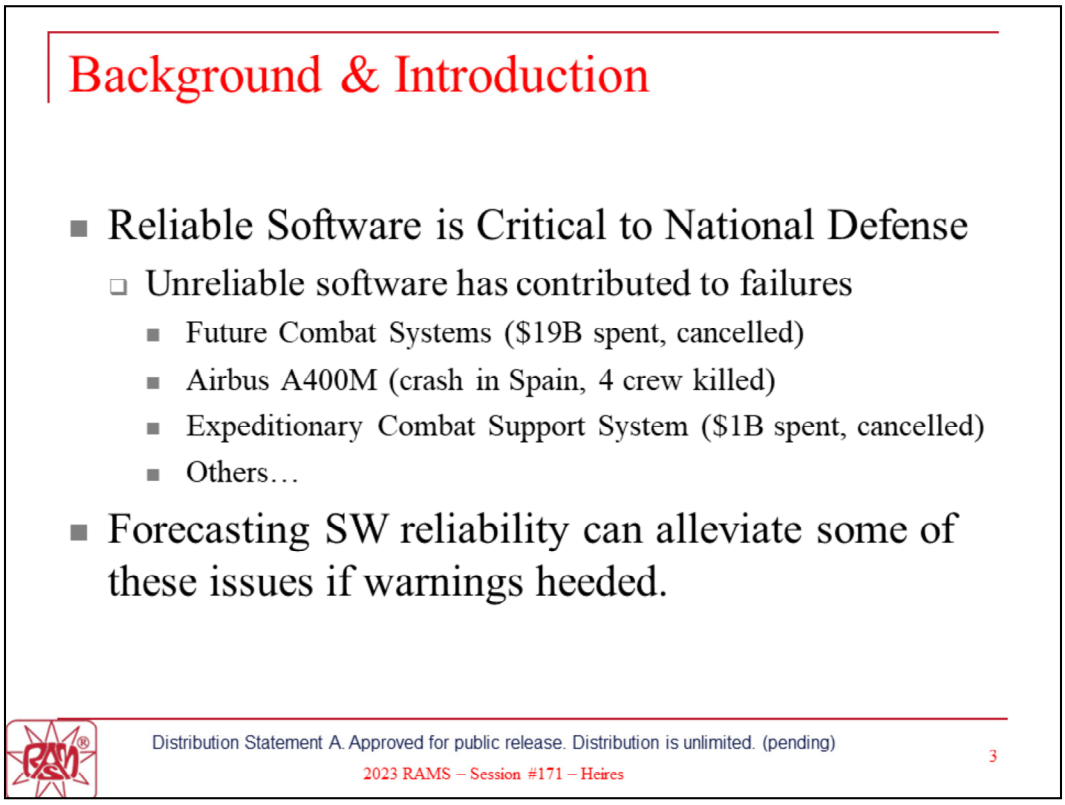
QSM, Inc.

On contract to USD(R&E)

The 69th Annual Reliability & Maintainability Symposium

Overview and Outline

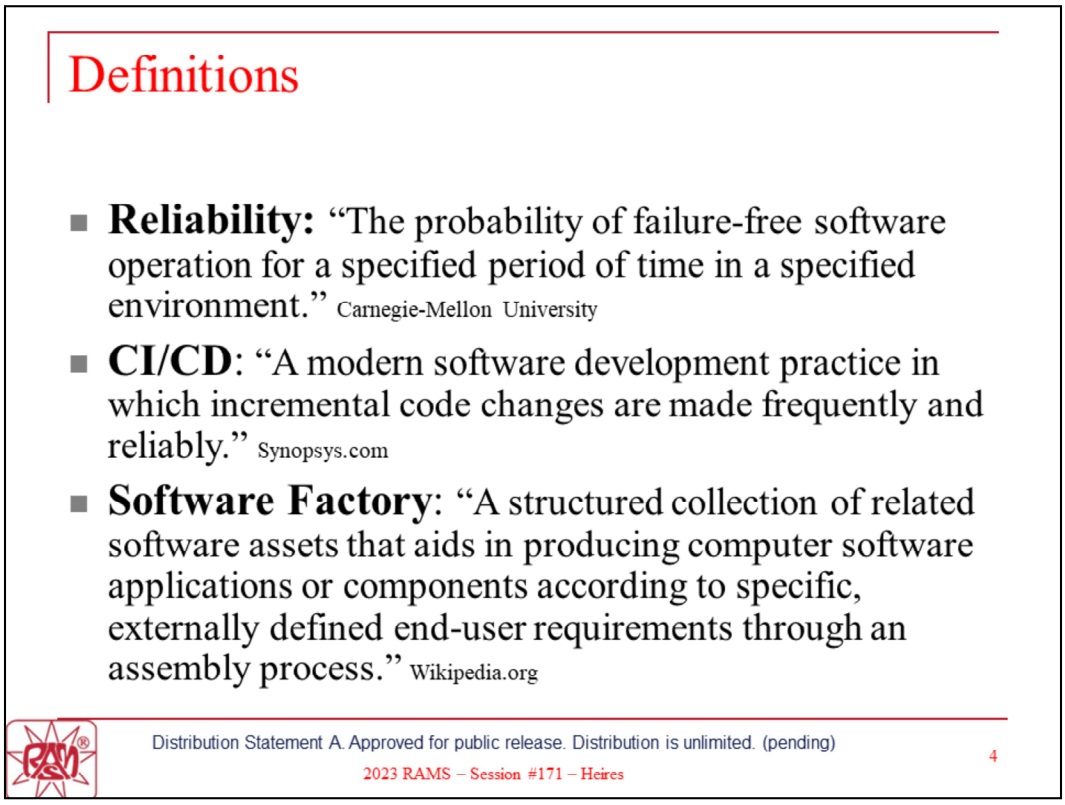
- Background & Introduction
- Definitions
- Challenges
- Modeling Basics



In the push towards Continuous Integration / Continuous Delivery (CI/CD) of military software engineering, time-to-market objectives have been supplemented with ongoing oversight and control. The Defense Science Board (DSB) recommended in their final report (Feb, 2018) that a "software factory" concept be emphasized in the source selection process. To support this concept and the digital transformation of reliability and maintainability (R&M), contractors will need to enable and begin to practice CI/CD to stay competitive on DoD development programs. Measurement, modeling and analysis techniques can be leveraged to ensure software development efforts meet reliability expectations while practicing CI/CD.

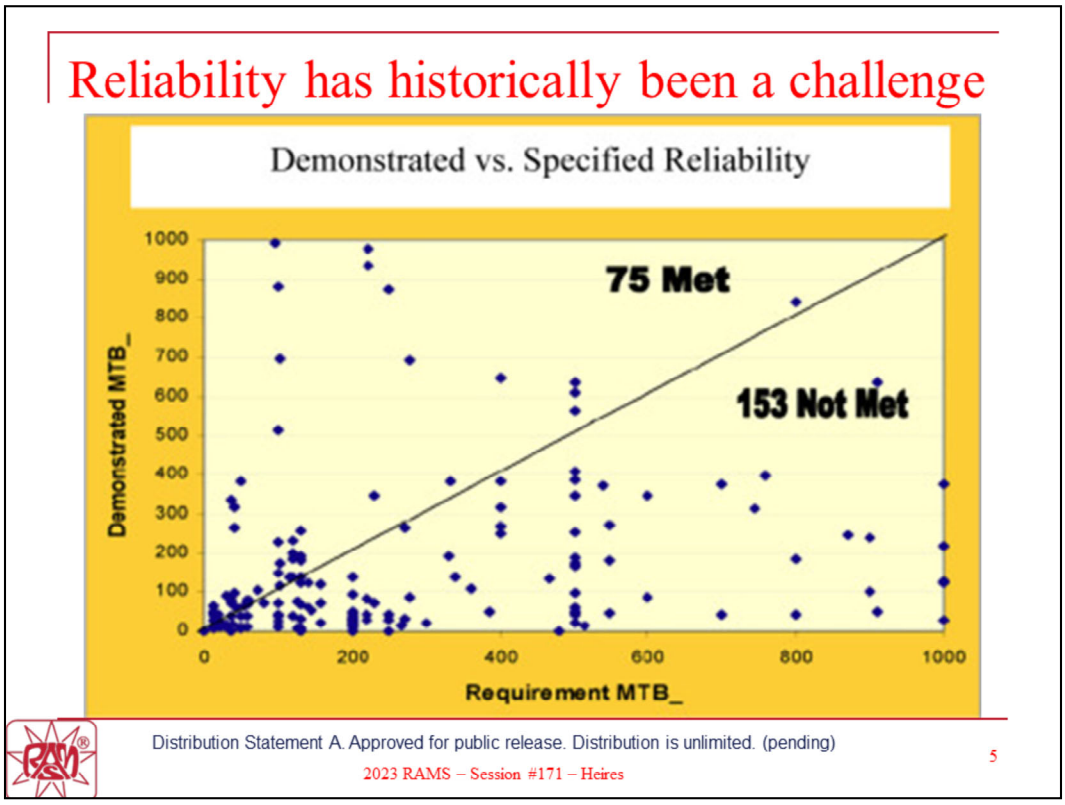
Program managers tend to think about the reliability of their software only when the customer finds defects in the field, causing delays and monetary impact. Because software defects are the fundamental drivers of software product reliability, it stands to reason that routine predictive modeling of software reliability would benefit all stakeholders.

Estimating and forecasting software reliability requires some specialized methods (e.g., curve-fitting, estimation models), and should be leveraged to explore the impact on software quality and/or reliability before a product development effort begins (or periodically throughout the development lifecycle). This paper explains some models in use and shares actual examples of DoD programs making use of this functionality to predict field reliability of software programs.



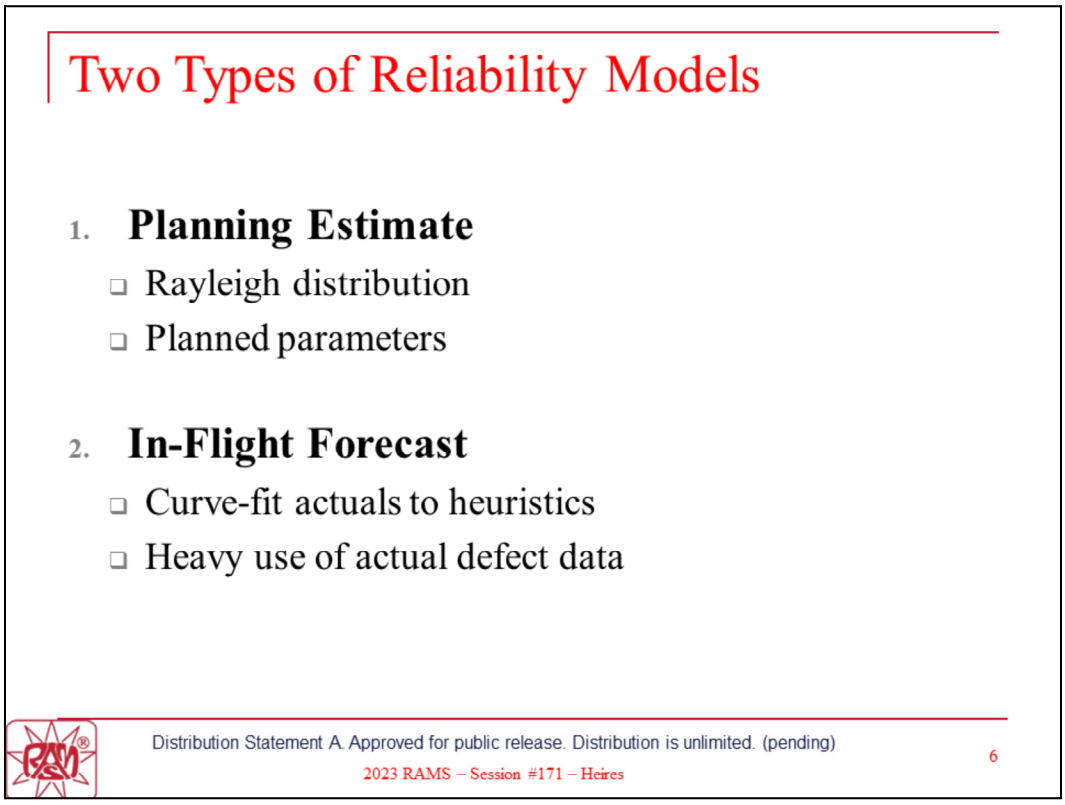
CI/CD Defined: "In very simple terms, CI is a modern software development practice in which incremental code changes are made frequently and reliably. Automated build-and-test steps triggered by CI ensure that code changes being merged into the repository are reliable. The code is then delivered quickly and seamlessly as a part of the CD process. In the software world, the CI/CD pipeline refers to the automation that enables incremental code changes from developers' desktops to be delivered quickly and reliably to production." Source: Synopsys.com

Software Factory Objectives: "Software factory–based application development addresses the problem of traditional application development where applications are developed and delivered without taking advantage of the knowledge gained and the assets produced from developing similar applications. Many approaches, such as training, documentation, and frameworks, are used to address this problem; however, using these approaches to consistently apply the valuable knowledge previously gained during development of multiple applications can be an inefficient and error-prone process. Software factories address this problem by encoding proven practices for developing a specific style of application within a package of integrated guidance that is easy for project teams to adopt. Developing applications using a suitable software factory can provide many benefits, such as improved productivity, quality and evolution capability." Source: Wikipedia.org



Software reliability is one of several characteristics which are knowingly or unknowingly traded off for time-to-market, cost, effort or functionality. All too often, reliability is short-changed.

Image Credit: Anton Petrus - stock.adobe.com



Rayleigh Model provides the time series projections for staffing, effort, cost, code construction and defects for a given estimation scenario. It is the optimal approach to match effort expenditure to the way the problem is ready to be solved. Rayleigh is one of the Weibull family of reliability functions.

Once a project is under way and the first defect counts begin to roll in, what is the most effective way to use that information? The best method will involve metrics that are easy to capture and interpret and allow the project to produce reliable and repeatable reliability forecasts. The methods outlined in this article have been in use for more than three decades and have worked well for organizations at all levels of process maturity and development capability.

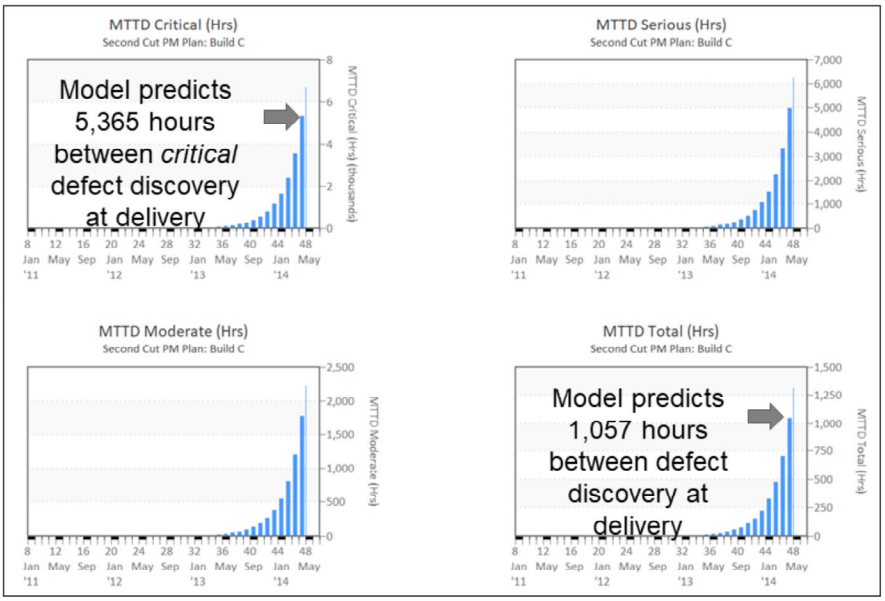
Defect prediction models can be broadly classified as either static or dynamic. Both have advantages and may be useful at various points in the lifecycle. Static models use final defect counts from completed projects to estimate the number of defects in future projects. Dynamic models use actual defect discovery rates over time (defects per week or month) from an ongoing project to forecast.

Research performed by Lawrence H. Putnam, Sr. (Putnam and Myers) shows that defect rates follow a predictable pattern over the project lifecycle. Initially, staffing is relatively low and few project tasks have been completed. Defect creation and discovery increase or decrease as a function of effort and work completion. As people are added to the project and the volume of completed code grows, the defect discovery rate rises to a peak and then declines as work tails off and the project approaches the desired reliability goals. This characteristic pattern is well described by the Weibull family of curves (which includes the Rayleigh model used in SLIM®)

Modeling Basics – Planning Estimate

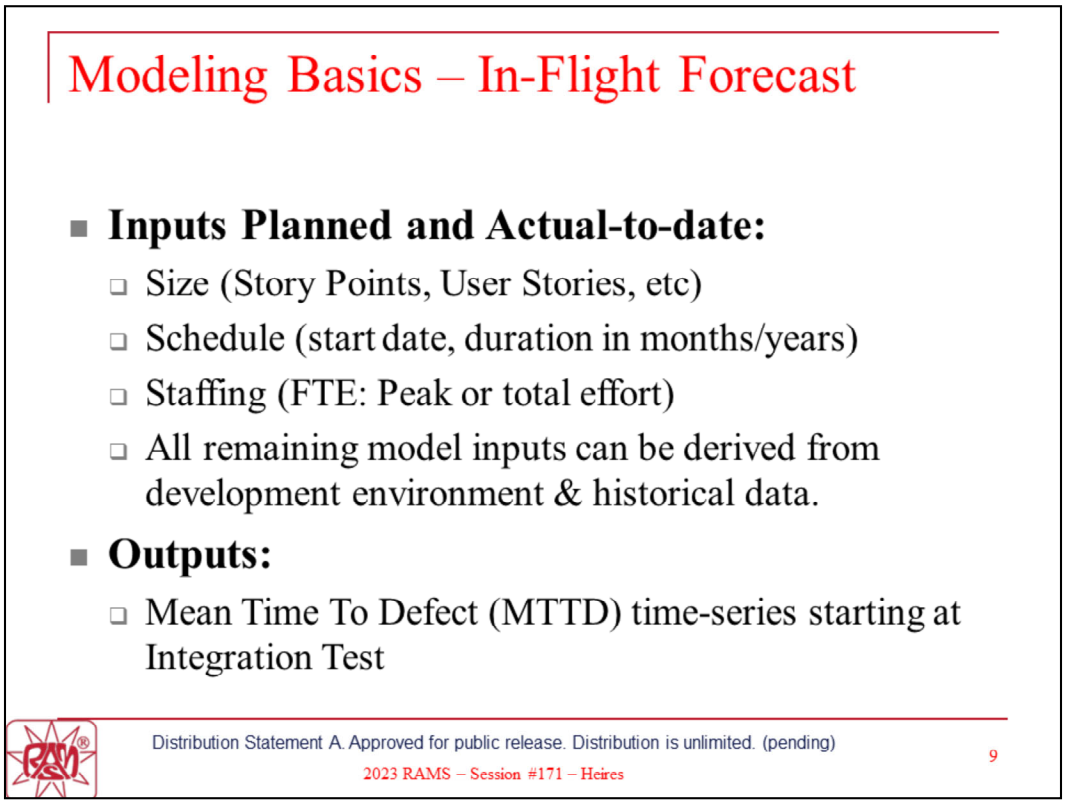
- **Inputs:**
  - Planned size (Story Points, User Stories, etc)
  - Planned schedule (start date, duration in months/years)
  - Available staffing (FTE: Peak or total effort)
  - All remaining model inputs can be derived from development environment & historical data.
- **Outputs:**
  - Mean Time To Defect (MTTD) time-series starting at Integration Test

Planning Estimate Outputs



Example from actual helo estimate (partial)

Build A_001 – Copy.sew (Second Cut PM Plan: Build C)

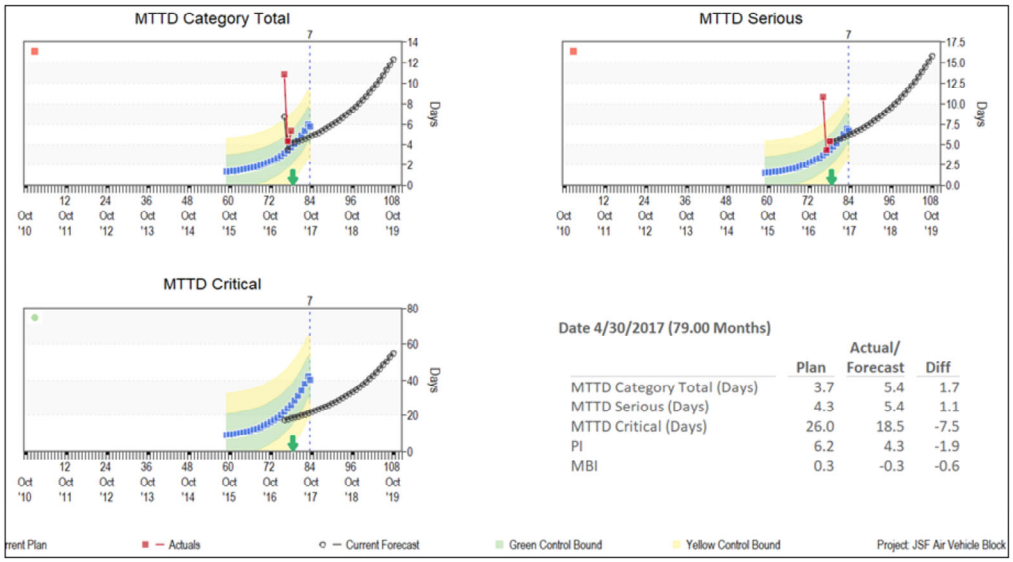


Defect rates have another useful aspect; they can be used to calculate the MTTD. MTTD is analogous to Mean Time to Failure. It measures reliability from the user's perspective at a given point in time, typically when the system is put into production for the first time. Though more complicated methods exist, it can be calculated quickly simply using the following formula (QSM, Inc.):

> *"To calculate MTTD, take the reciprocal of the number of defects during this month and multiply by 4.333 (weeks per month) and the days per week for the operational environment. For example, if there were five errors during the first month of operation and the system runs seven days per week, the average MTTD value would be (1/5) * 4.333 * 7 = 6.07 days between defects. If there are no defects in the first month, the MTTD in the first month cannot be calculated."*

MTTD makes it possible to compare the average time between defect discoveries to the software's required mission profile and predict when the software will be reliable enough to be put into production. Mission-critical or high-reliability software should have a higher Mean Time to Defect than a typical IT application. Software that must run 24 hours a day and seven days a week requires a higher Mean Time to Defect than software that is only used for eight hours a day from Monday to Friday. MTTD considers all these factors explicitly.

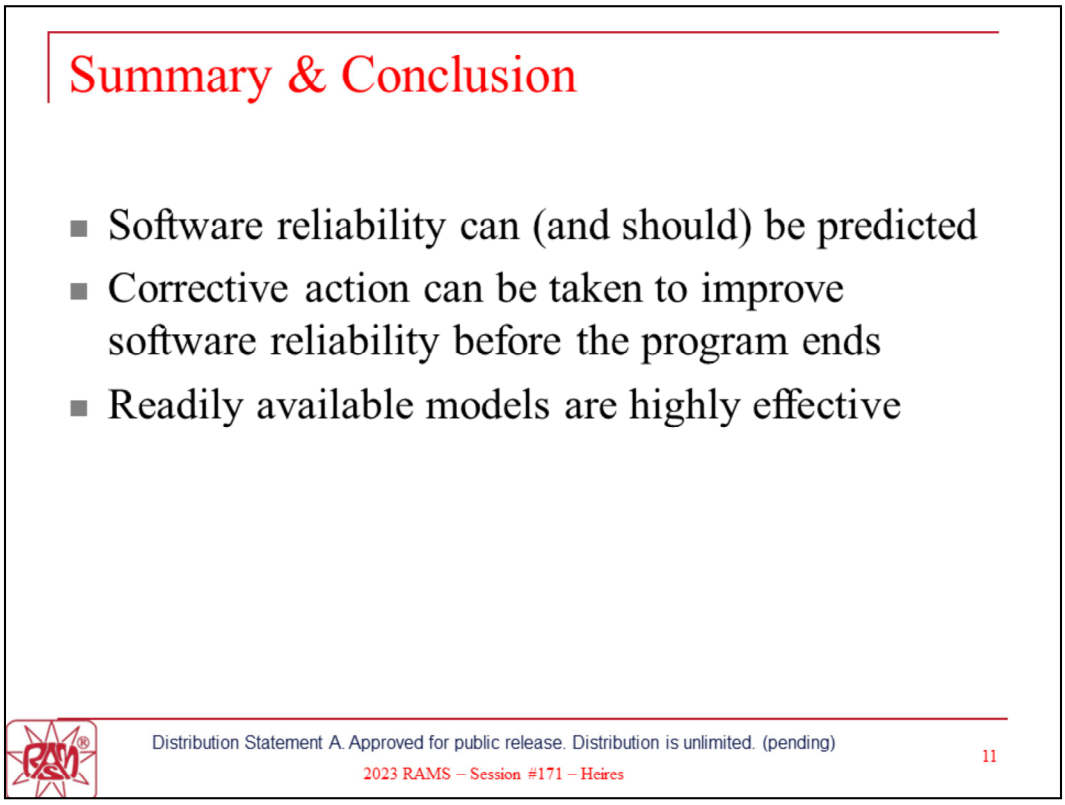
In-Flight Forecast Outputs (monthly data)



Date 4/30/2017 (79.00 Months)

	Plan	Actual/ Forecast	Diff
MTTD Category Total (Days)	3.7	5.4	1.7
MTTD Serious (Days)	4.3	5.4	1.1
MTTD Critical (Days)	26.0	18.5	-7.5
PI	6.2	4.3	-1.9
MBI	0.3	-0.3	-0.6

Example from actual TBD estimate (partial)



MATCHING RELIABILITY STANDARDS TO THE MISSION PROFILE

How should organizations determine the right reliability standard for each project? A good place to start is by asking, “What does the software do?” and “How reliable do we need it to be?” Defect rates, taken in isolation, aren’t terribly helpful in this regard. Software developers need to know how long the software should run in production before users encounter defects of various severities.

MTTD can be customized to reflect the project’s unique mission profile. The mission profile, in turn, can depend on a variety of factors. Required reliability for onboard navigation software aboard a fighter jet may depend on how long it can stay in the air before refueling. For other types of software, human considerations like time on shift or time until a unit is relieved determine the required reliability. Different types of software will have different mission profiles. A flight management system may be operational 24 hours per day, seven days per week, 60 minutes per hour, and 60 seconds per minute. A billing application for a doctor’s office, on the other hand, may only be required to operate eight hours a day, five days a week. Because the flight system operates continuously for a longer period of time, it requires a higher reliability (or MTTD). 39

Finally, MTTD can be calculated for total defects or customized to reflect only high-priority defects. End users may not care how long the system runs between cosmetic errors but for mission-critical applications, increasing the average time between serious and critical errors can literally mean the difference between life and death.

Next Steps and Future Work

- Questions to or from the audience
- Model outputs courtesy QSM, Inc.