Abstract: Space Shuttle Software Development and Certification

Man-rated software, "software which is in control of systems and environments upon which human life is critically dependent," must be highly reliable. The Space Shuttle Primary Avionics Software System is an excellent example of such a software system. Lessons learn from more than 20 years of effort have identified basic elements that must be present to achieve this high degree of reliability. The elements include rigorous application of appropriate software development processes, use of trusted tools to support those processes, quantitative process management, and defect elimination and prevention. This presentation highlights methods used within the Space Shuttle project and raises questions that must be addressed to provide similar success in a cost effective manner on future long-term projects where key application development tools are COTS rather than internally developed custom application development tools.

Space Shuttle Software Development and Certification

James K. Orr

Presentation Developer: Johnnie A. Henderson
Space Shuttle Orbiter Avionics Software

October 12, 2000

Project Overview

- **Project and process evolution began in mid-70's**
- **Software exhibits world-class quality and reliability for life-critical operations**
  - Very high maturity software process (NASA HQ assessed as SEI Level 5)
  - ISO 9001 registered process
  - 19 years of demonstrated flight safety and ultra-reliability
- **Onboard software controls all phases of Shuttle flights (manual and auto)**
- **Primary Avionics Software System (PASS)**
  - Provides automatic and fly-by-wire control of critical shuttle systems which executes in redundant computers
  - 430,000 SLOC primarily in HAL/S
- **Support Software (1,400,000 SLOC)**
  - Software integration automation
  - Full fidelity 6-degree of freedom simulator
  - Configuration control and process control tools
  - Quality and process metrics data bases

Principles of Providing High Reliability Software

- Safety certification is currently based on *process adherence* rather than *product*.
- Assumption is that a *known, controlled, repeatable* process will result in a product of known quality.
- Use "trusted" tools to develop, build, release and maintain the software.
- Use measurements to continuously assess the health of both the process and the product.
- Relationship between quality and reliability must be established for each software version and statistically demonstrated for the required operational profiles.
- Quality must be *built into* the software, at a *known* level, rather than *adding* the quality after development.

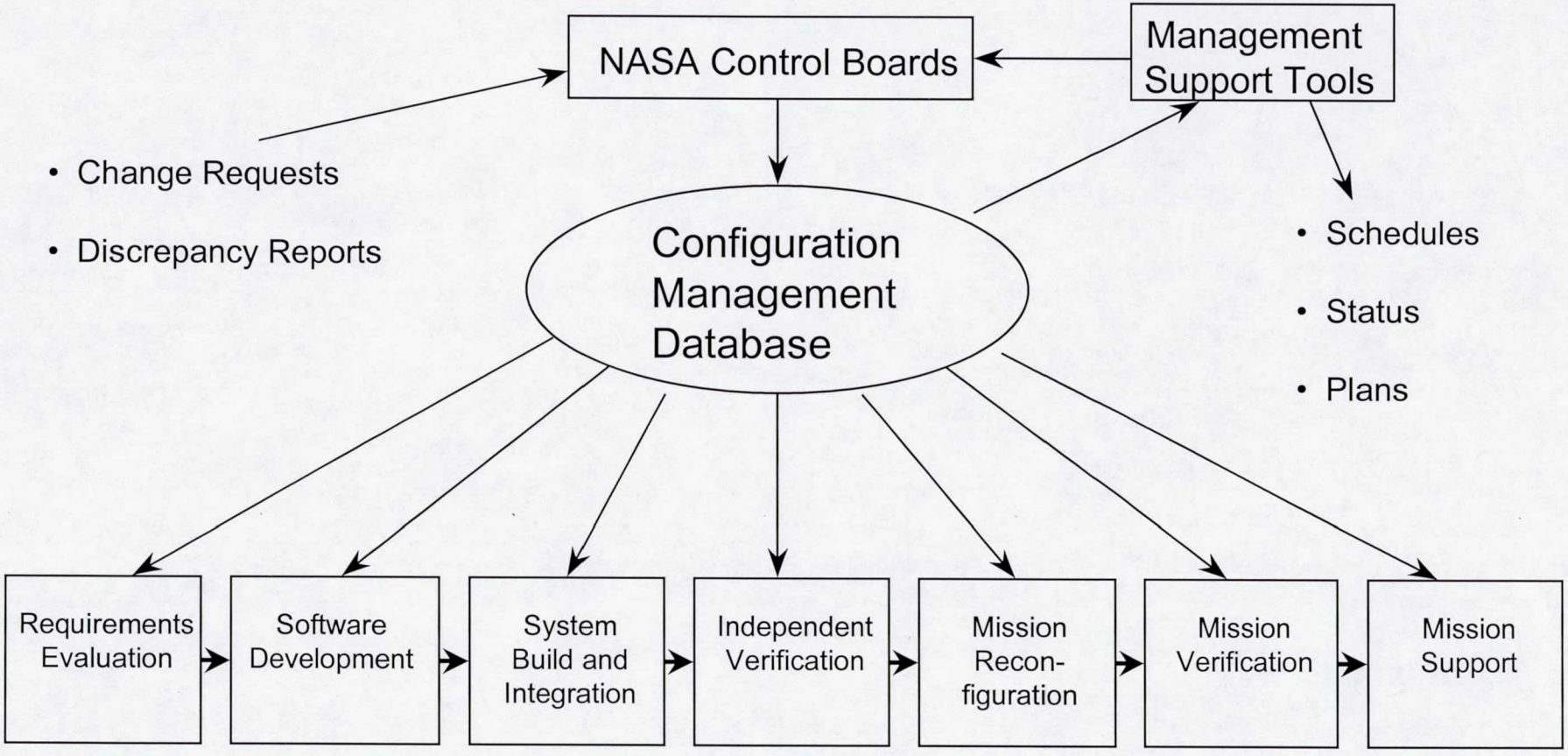
Space Shuttle Software Development Philosophy

- **Goals:**
  - Meet letter and intent of customer requirements
  - Perform in accordance with customer expectations
  - Error-free software
- **Project Organization Function:**
  - Requirements analysis
  - Software development (design, code, test)
  - System build and integration
  - Independent verification
  - Reconfiguration/certification
  - Customer and field support
- **Configuration Management**
  - Formal board structure
  - Documentation of requirements and design baselines
  - Stored in Configuration Management Database

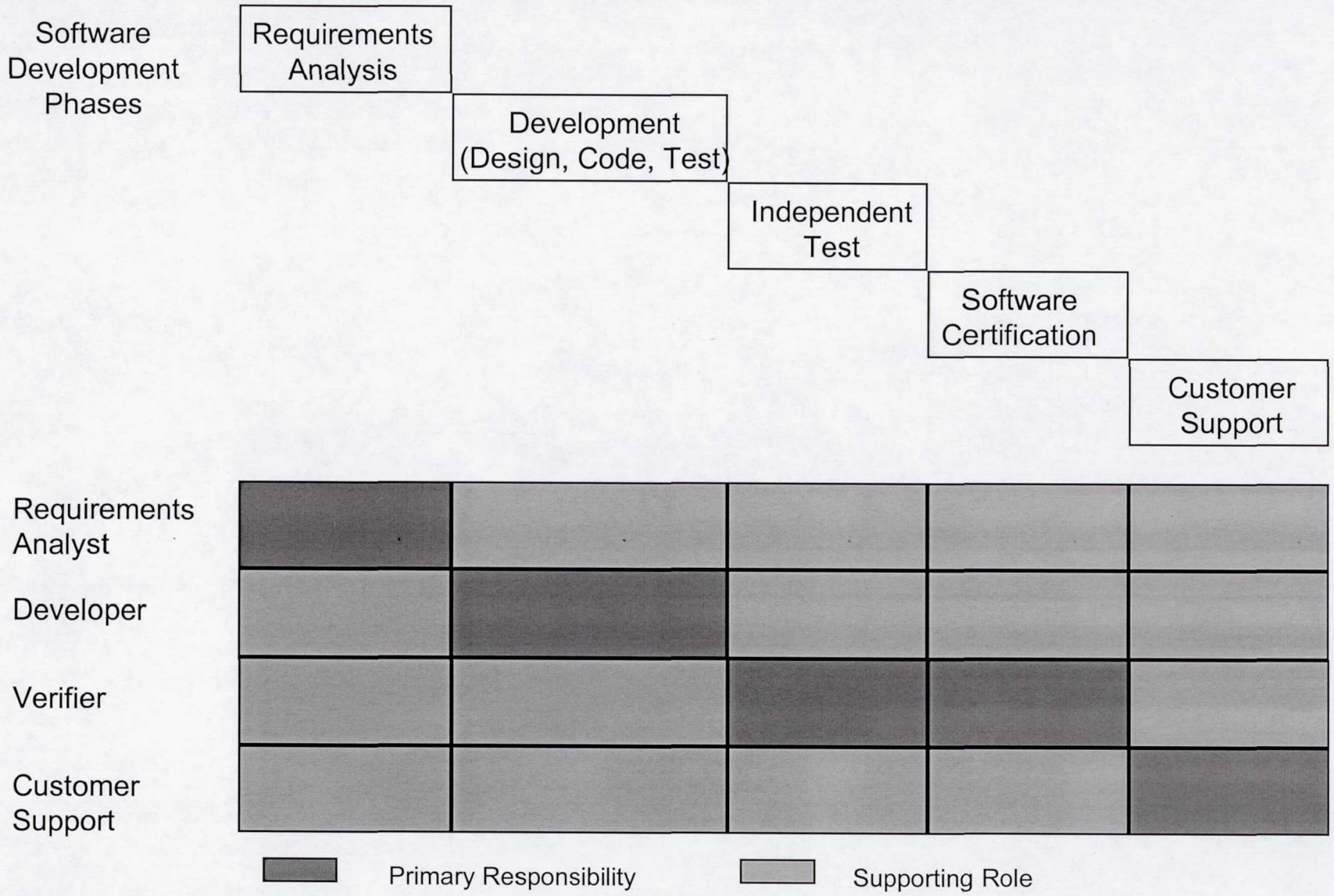
Configuration Management

- **Ensures delivery of no more or less than what is required**
- **Spans the software life cycle**
  - Begins at requirements definition, continues until product retirement
  - Extend to baselines, software, supporting tools, testing, documentation
- **Key tool is Configuration Management Data Base**
  - Documents schedules and delivery content
  - Provides extensive information about requirements and defects
  - Controls software build process, software build contents
- **Traceablity of requirements to code documented**
  - Requirements tied to source statement reference numbers in the module prologue
  - Requirements mapped to test cases, documented in test case prologue
- **Control board structure mirrors customer board structure**
  - Perform project planning and integration
  - Provides extensive communication of status

Shuttle Software Configuration Management

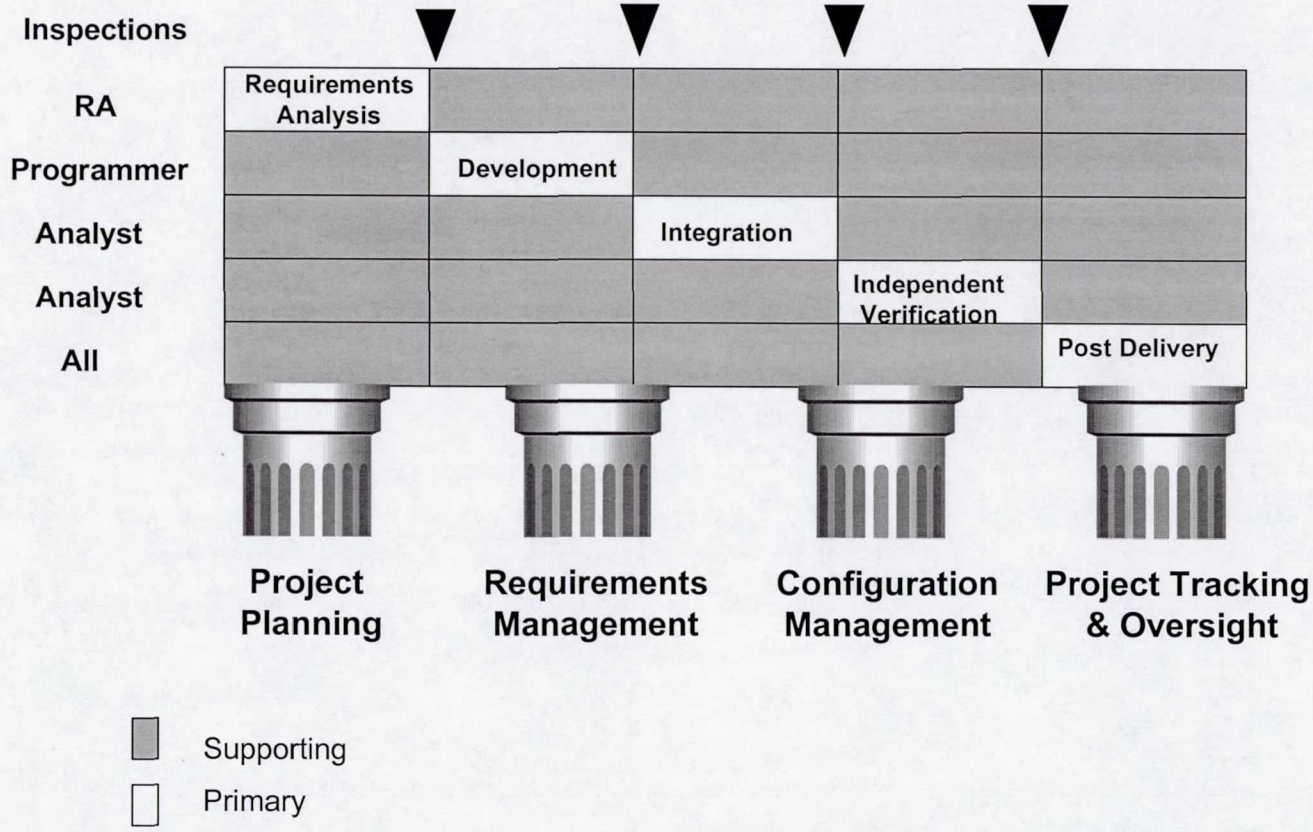


Roles in the Software Development Process



Inspection Points in the Life Cycle

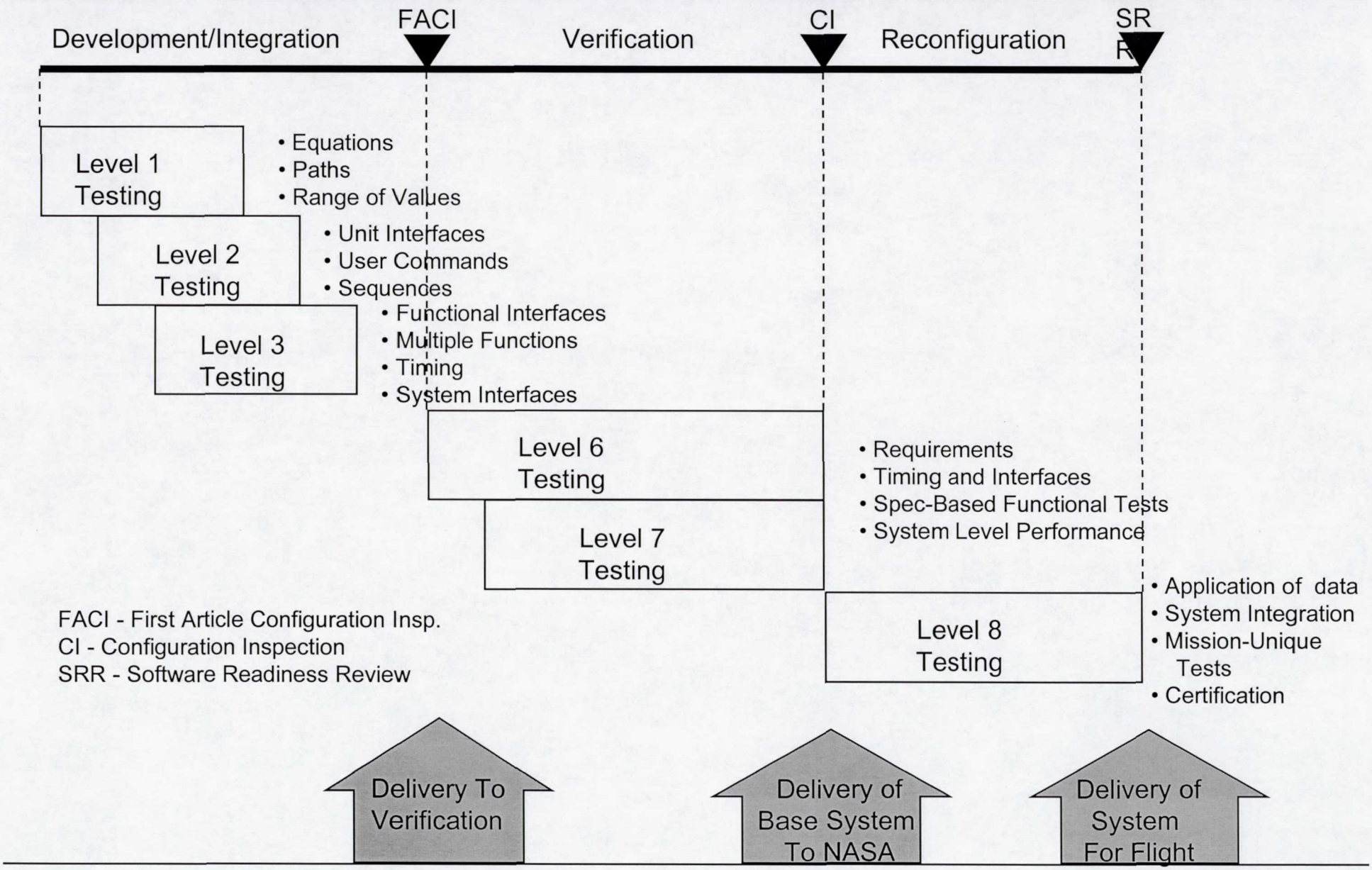
APPLYING INSPECTIONS



Inspection Points in the Life Cycle

- **Inspections used in all phases of software development and test**
- **At each process step, inserted defects can be found**
- **Software defects are inserted**
  - Requirements definition
  - Design
  - Code
- **Software defects are found**
  - Inspections
  - Development test (unit and functional)
  - Independent verification
- **Group inspections eliminate defects more effectively than testing**
- **Inspection data provide historical basis for statistical analysis and quality management**

Shuttle Software Test Approach



Independent Software Verification

- **Aspects of independent verification permeate the software development process**
  - Requirements analyst often a system tester
  - Inspection teams (requirements, design, code) include members from all life cycle phases
- **Emphasis on defect prevention**
  - Defect cause analysis
  - Latent error analysis
  - Trend analysis, feedback
- **Software process includes many verification levels**
  - Tailored to quality, resource requirements
  - Verification philosophy assumes software is untested when received
  - Verification plans developed focused on the requirements

Other Tasks

- **Testers do more than test**
  - Nearly half of verification-discovered defects are found through static analysis
    - Manual desk checks
    - Analysis tools
    - Many defects would have escaped detection through traditional testing
  - Participate in formal inspections
    - Requirements inspections
    - Design and code inspections
  - Hold verification interface meetings on complex changes
  - Participate in defect cause analysis
  - Participate in product audits

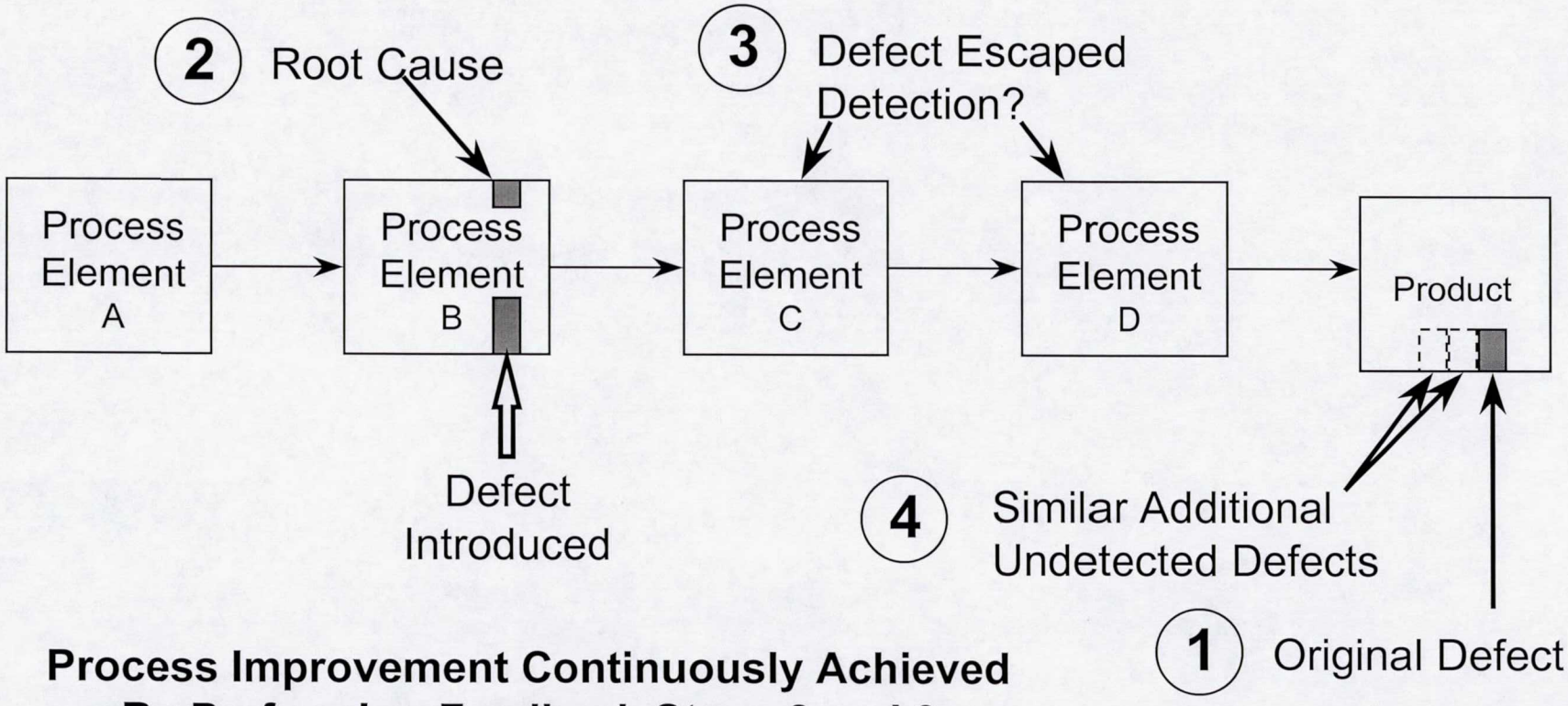
Philosophy

- **Focus on defect prevention**
  - Build quality in versus testing errors out
  - Use defect data to drive process changes
  - For every defect, ask how process can be changed to prevent it next time
- **Improve process and product quality**
  - Measure quality of both the process and product
  - Measure impact of process changes on quality
- **Depend on defects to achieve perfection**
  - Manual process not 100% repeatable
  - Makes perfect process or product impossible
- **High quality achieved through a sequence of process corrections**

Defect Elimination Process

Steps performed

1. Remove defect
2. Remove root cause of defect
3. Eliminate process escape deficiency
4. Search/analyze product for other, similar escapes

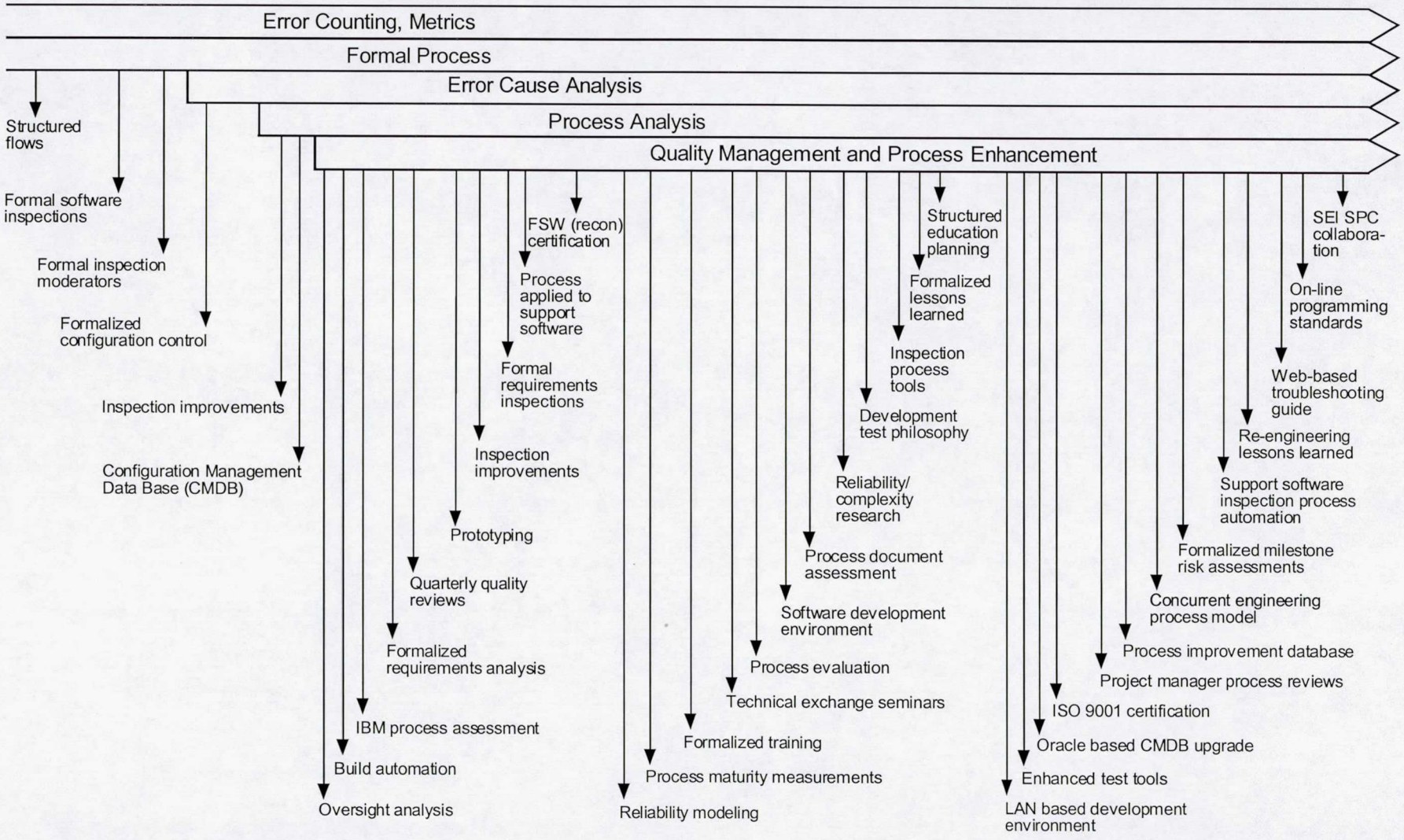


Process Improvement Continuously Achieved By Performing Feedback Steps 2 and 3

Foundation of a Mature Process

- **A built-in part of a mature process is to change the process…assuring continuous improvement.**
  - Changes based on detailed *introspective* and *metric* analysis for process *optimization*
  - Process under statistical *management* assuring continuous improvement
  - Measure quality and reliability for consistent elements of the software
    - Do not treat a software version as a set of homogeneous lines of code (age, pedigree)
  - Model, predict, and compare to actual quality and reliability
  - Use active feedback of process metrics to *control* fault density
- **Bottom line – rigorous processes + "trusted" tools + measurements = quality software**

Continuous Process Improvement



Groups_graphics_pubs_PASS_FSW_001.cv5

Recommendations for DfS Consideration

- **Focus on *total* life cycle, not just development through initial certification**
  - Program life on order of 10 to 30 years
  - Conceive and manage for an evolving infrastructure
  - Have the ability to make small, late changes within a reasonable elapsed time
- **Rigorous processes, "trusted" tools and in-process measurements all necessary to allow development of software of known reliability**
  - New releases of development tools may force extensive regression testing of deployed applications if applications developer has no insight into areas of changes and quality of the new releases
  - Specific version of vendors tools may become unsupported because of vendors' overall marketing strategy
  - Application developer needs to be able to assess significance of vendor product problems and their effect on the application software to know if PRACA reporting is required
- **Propose a model to allow "certification" of vendors process and products via an independent third party (similar to ISO and SEI CMM)**
  - Establish non-disclosure agreements with selected vendors
  - Perform process assessments to insure mature, repeatable, rigorous processes in place

Recommendation (continued)

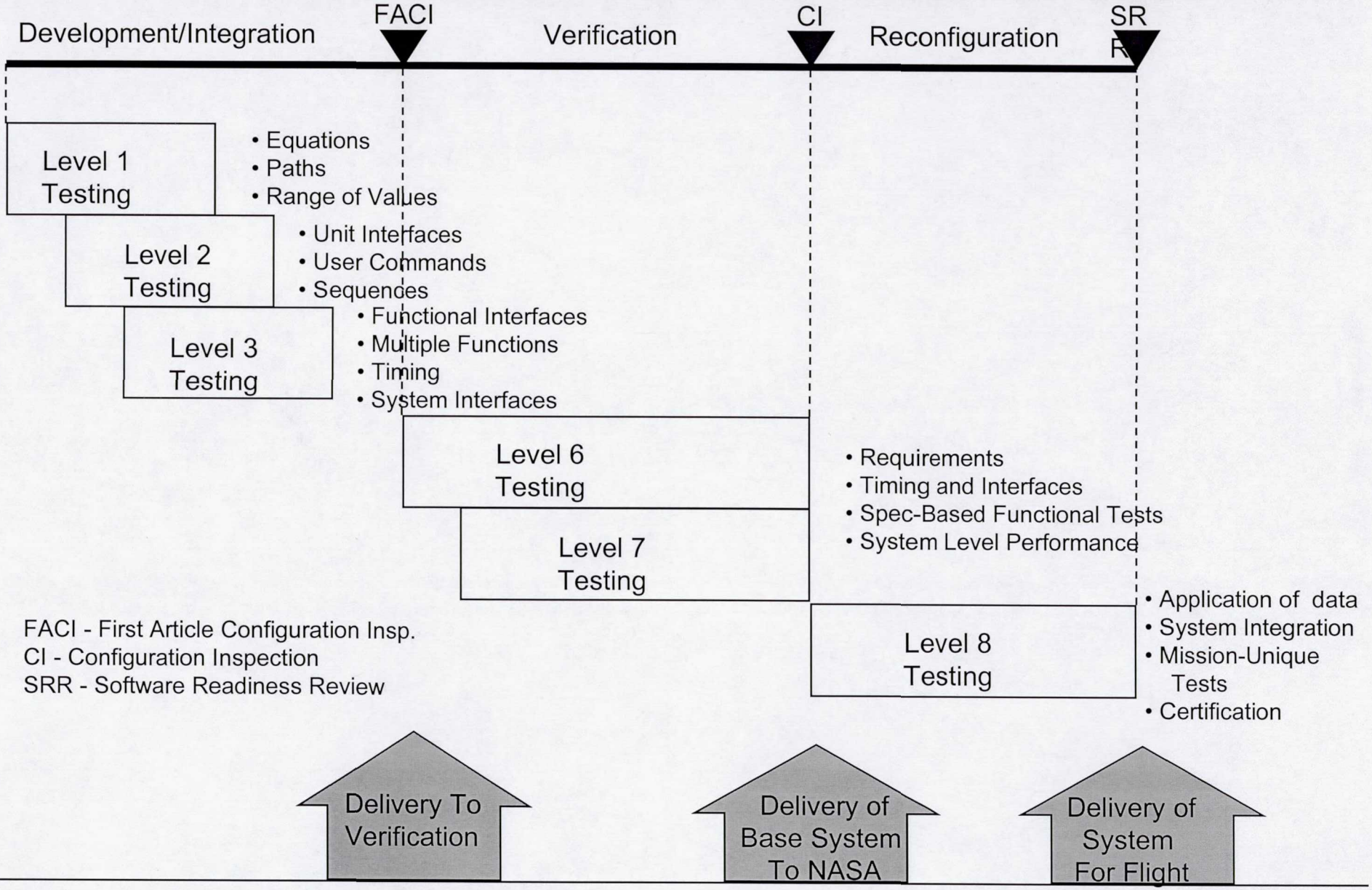
- Third-part assessor provides to subscribing applications developers information on:
  - Areas of tools undergoing changes
  - Scope of changes (including problem report correction, interface changes, or new functionality)
  - Predictions of quality and reliability of new releases based on monitoring vendor's in-process measurements

- **Business Model**
  - Independent third party paid by vendor to perform process assessment
    - Process assessment replaces ISO certification and independent SEI CMM assessments, thereby resulting in neutral net cost increase
  - Application developers pay independent third party for reports on new release content and quality/reliability by function
    - Precise information on areas changed can be used to reduce, or at least focus, regression testing when stepping up to new releases resulting in net cost savings
  - Application developers pay for "alert" notification service – identification of significant problems in vendor's product used as a "trusted" tool
    - Allows application developer to initiate preventive or correction action if "trusted" tool problem could affect developer's application
    - Allows application developer to maintain deployed applications at a lower cost for required level of safety/reliability

Attachments

Shuttle Software Test Approach

Shuttle Software Test Approach



Unit Testing (Level 1)

- **Classic white box testing**
  - Test the code against the design
  - Done with requirements knowledge
- **Focuses on individual code units, ignores interfaces**
- **Verifies code unit against design**
  - Logic
  - Data handling
  - Design extremes and constraints
- **Typical test coverage defined is statement, branch or condition**
- **No longer done in this form for shuttle software due to low return on investment**

Subsystem Testing (Level 2)

- **A *light gray* - stronger emphasis on requirements, customer usage scenarios, with detailed insight into code implementation**
- **Verifies interfaces within a specific component or function composed of integrated modules/units**
  - Testing of all internal and external interfaces
  - Data and error handling
- **Can be difficult to accomplish in complex system where requirement or change may affect several modules**
- **Off-nominal scenarios added**
- **Regression tests included as appropriate**
- **Test for proper operation of functions/components against requirements**
- **Test specification and results inspected with no independent verification personnel present**

Integration Test (Level 3)

- Totally black box testing
- Objective is to test the health of the entire system following integration of new/changed code
- Verifies the ability of a subsystem to execute its functions in parallel or serially (where output of one function is input to another function)
- Creates the *checkpoints* used to begin Level 6 or 7 testing in the middle of complex operational scenarios
- Tests are independent of most system changes and reused.

Verification Testing

Independent Verification (Level 6)

- **Detail/Functional Testing (Level 6)**
  - Detail Testing – Explicit verification of all requirements:
    - Logic paths, algorithms, decision blocks, variable pools
    - Traceability provided between specific test cases and requirements paragraphs
  - Functional Testing – Explicit verification of the following:
    - Internal and external data interfaces (among software modules, I/O)
    - Sequencing of events
    - Order of module execution
    - Proper operation of overall subsystems being tested (e.g., Guidance, Navigation, Flight Control)
- **Test planning/Preparation**
  - Level 6 Verification Test Procedures inspected at formal meetings with peer to which customer and members of technical community are invited; test cases and results reviewed internally and by customer

Detailed Verification

- **Organization structure**
  - Separate from development organization
  - Adversarial relationship with the development organization
  - Emphasis is on defect detection
  - Major participants in development inspections - allows verification to contribute to defect prevention
- **Detailed verification philosophy**
  - Despite Level 1, 2, and 3 testing, software is considered untested
  - All software changes are tested using test plans independently generated by the verification analysts
  - Delta code inspection done to assure that all code changes are authorized

Detailed Verification

- **Detailed/functional testing**
  - Gray box testing - all requirements tested, but software implementation is considered in the generation of test cases
    - Algorithms, statements, conditions - similar to unit testing
    - Functional testing of all internal and external interfaces
    - Sequencing of events
    - Order of module execution
  - Proper operation of overall subsystems being tested

Detailed Verification

- **Test case generation methods**
  - Test cases mapped to requirements
    - Fundamental activity for black box testing
    - Objective is 100% test coverage of requirements changes - methods used are unique to each analyst
    - Insight is gained from participation in the requirements inspection
  - Design and code are analyzed
    - Objective is enhancement of the test plan developed from the requirements analysis and mapping
    - Total coverage of changed code required
  - Off-nominal scenarios added
  - Regression testing included

Detailed Verification

- **Epilogue generation**
  - Written for each test case to document test results - important information for reuse or if audit is required
    - What was tested
    - Who tested
    - Deviations from expected results
    - Data set names
  - Input to the inspection with the test results

Detailed Verification

- **Test plan inspections**
  - Verification test plan is written to document test details - results from requirements tracing, design and code analysis, scenario identification, and regression test plans
  - Attended by a peer analyst, customer, sometimes the requirements author or shuttle community experts, and a requirements analyst
  - Required for most tests - can be waived by the customer
  - Checklist used
  - Formal actions are documented and tracked

Detailed Verification

- **Peer review of test results**
  - All test cases are reviewed by a peer analyst
    - Intended to detect oversights or omissions before customer inspections
  - Focus is on common oversights and satisfaction of the test plan
  - Formal checklist is used
- **Customer review of test results**
  - Attended by the test analyst and a customer analyst - other non-USA expert may be invited
  - Issues are formally documented and tracked

Independent Verification (Level 7)

- **Performance Testing (Level 7)**
  - Designed to reveal overall system responses and verify system performance
  - Off-nominal operations
    - "Reasonable or expected" failures and anomalies due to hardware, environment and user error
    - "Extreme or catastrophic" stress
      - Low probability failures
      - To software breakpoint
      - To integrated system breakpoint
- **Test Planning/Preparation**
  - Level 7 performance test plans and procedures formally presented to customer, technical community for approval, and results formally reviewed by customer and technical community

Performance Verification (Level 7)

- **Totally black box testing**
- **Totally independent of Development**
- **Objective is to verify overall system performance**
- **Not all changes are tested - must have a visible effect on performance or to the user**
- **System integrity tests**
  - Test normal flight critical and abort modes
  - Classic regression tests
- **Capability tests**
  - Selected to evaluate the effects of specific software changes on the overall performance of the avionics systems
  - Off-nominal operations - reasonable or expected failures due to hardware, environment, or user error
  - Extreme or catastrophic stress

Performance Verification

- **Test planning**
  - Formal test planning meetings held by the test team
    - Internal USA meeting
    - Analysts present recommendations to test or not test each software change
    - Group decision on which changes will be included
  - Performance testers are also requirements analysts and therefore have excellent insight into the changes being considered
  - Selected changes are formally documented in a test plan
  - Test plan is reviewed by the customer and by the shuttle technical community
    - Tests may be added based on feedback from this review
  - Test plan is placed under configuration control

Performance Verification

- **Test results analysis**
  - Specialized plots generated to illustrate shuttle performance using the software
  - Numerical data examined to verify navigation and guidance algorithms
  - Formal performance test review packages are delivered to the customer
  - Technical reviews are held for each test case with the customer and with the shuttle technical community
  - Issues are tracked formally to completion
  - Formal performance test review is held with the shuttle technical community

Performance Verification

- **System Software performance**
  - Emphasis is on regression testing and timing measurements
  - Tests are run for critical flight phases and abort modes
  - Black box
  - Measurements include I/O and CPU performance but also many others
  - Measurements are tailored to demonstrate the continued acceptable performance of the system software architecture
    - Performance of the I/O profile in relation to the completion of necessary computation
    - Maintaining synchronization of multiple CPUs

Performance Verification

- **Reviews for system performance testing**
  - Test cases are reused on every system, reducing the need for inspection at the front end - minor tuning is sometimes necessary
  - Test case setup changes, normally minor, are reviewed by peer analysts
  - Test case results require the same level of review as application performance tests
    - Formal test report
    - Technical review with the shuttle community

Quality Measurement

Metrics and Interpretation

- **Early Detection**
  - What percent of all errors made were found and eliminated before being implemented into the new version?
- **Process Error Rate**
  - How many errors were present in the new version but were found before formal delivery?
- **Product Error Rate**
  - How many errors did our process miss? (Still present at formal delivery)

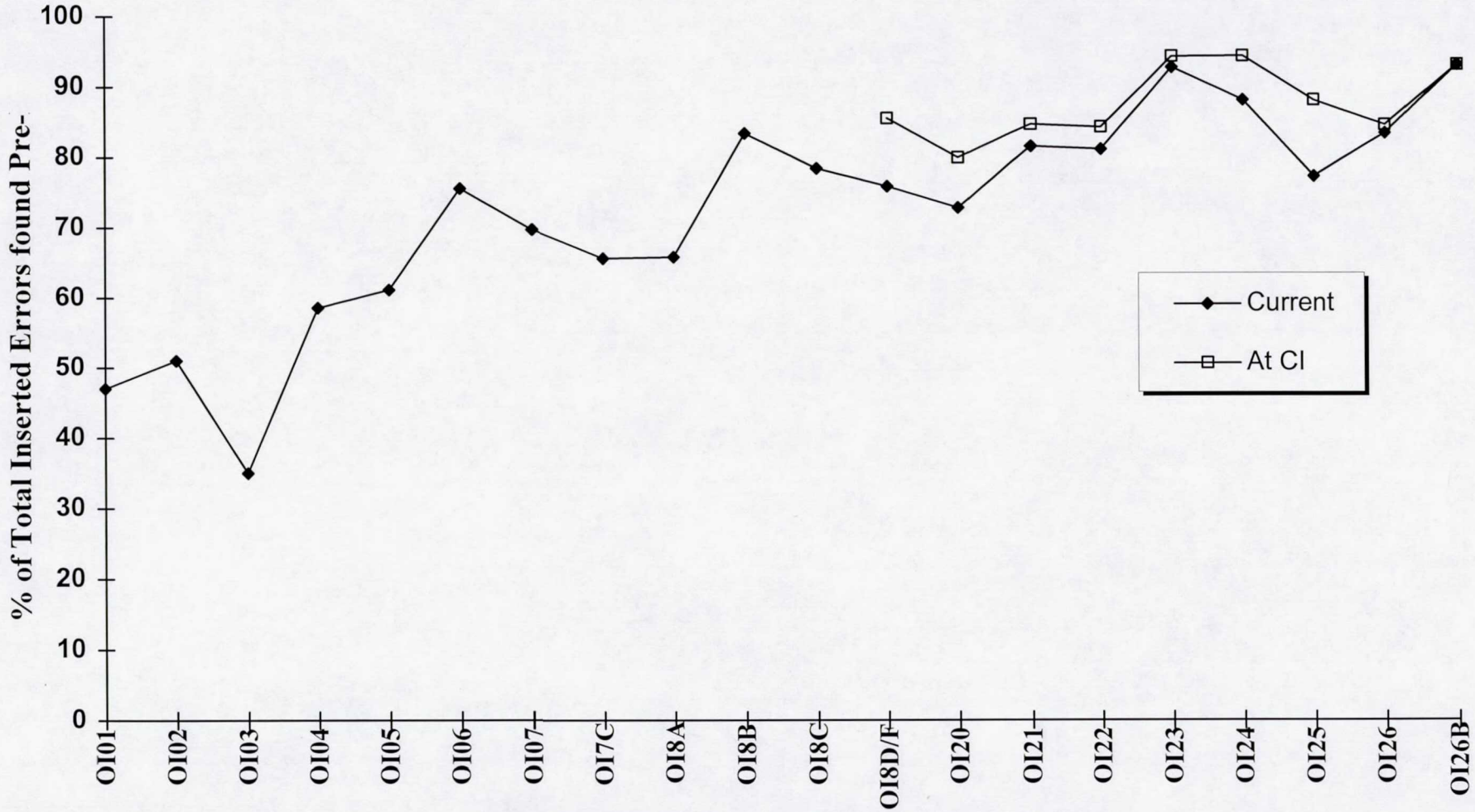
Early Detection

$$\frac{\text{\# of Pre-Build Major Errors* X 100}}{\text{\# of Pre-Build Major Errors + Valid DRs}} = \text{Early Detection \%}$$

- **Early Detection has several advantages**
  - Errors found early cost less to fix
  - Errors removed before build do not hinder verification process

* Pre-Build Major Error is an error which would be a DR if it were on an already built system.

Early Detection



CI26B081.XLS

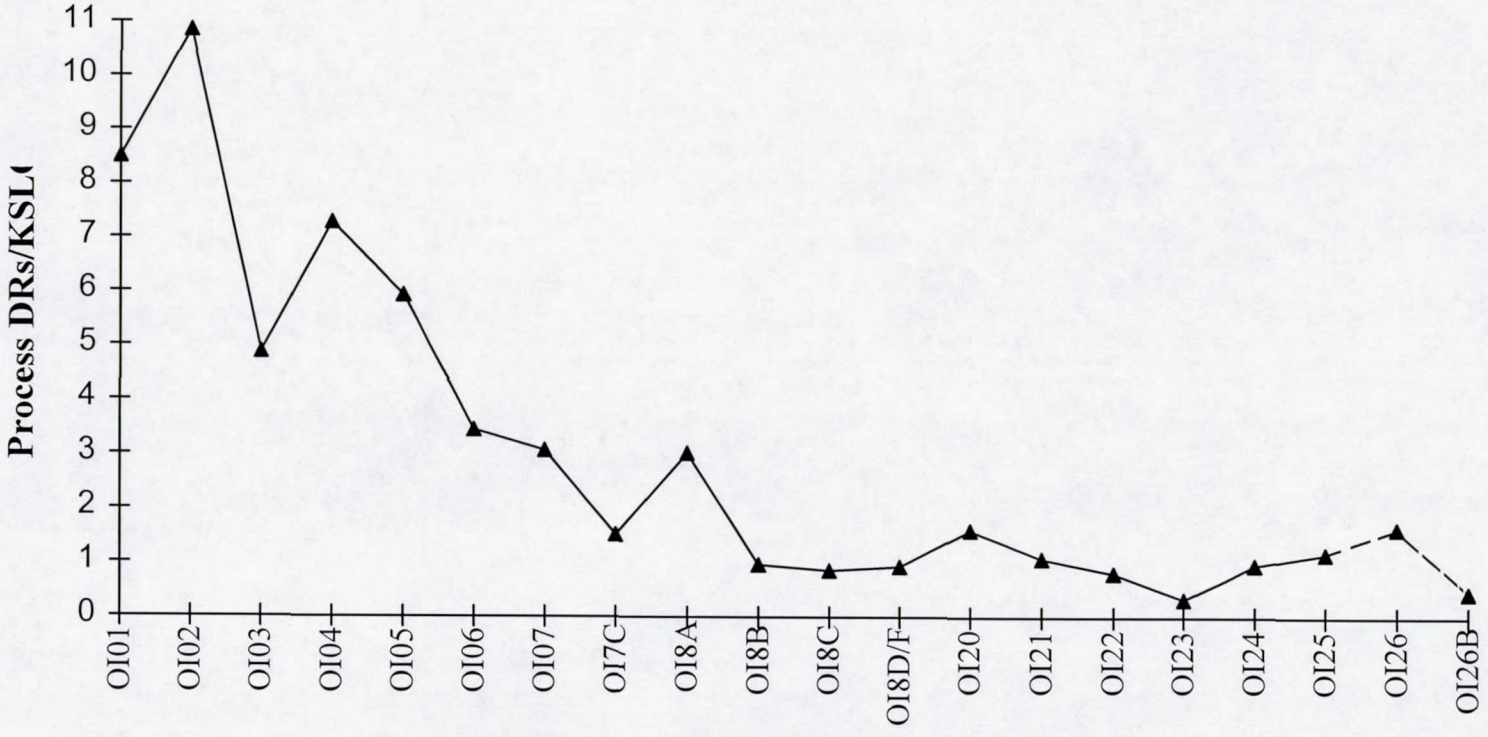
Process Error Rate

$$\frac{\text{\# of Valid DRs during Development Cycle}}{\text{New/Changed KSLOCs}} = \text{Process Error Rate}$$

- **Process Error Rate measures the number of DRs discovered during the development cycle between the time the code becomes part of a built system and the time that system is first certified for flight**
  - OI7C and OI8A measured to CI
  - All remaining OI systems measured to the first SRR for a flight off that OI
- **The Process Error Rate gives an indication of the quality of the flight software before use in a shuttle mission.**
  - Measure of how many errors were made in the flight software but were found before flight.

Process Error Rate

BUILD TO FIRST SRR
Based on new/changed SLOCs



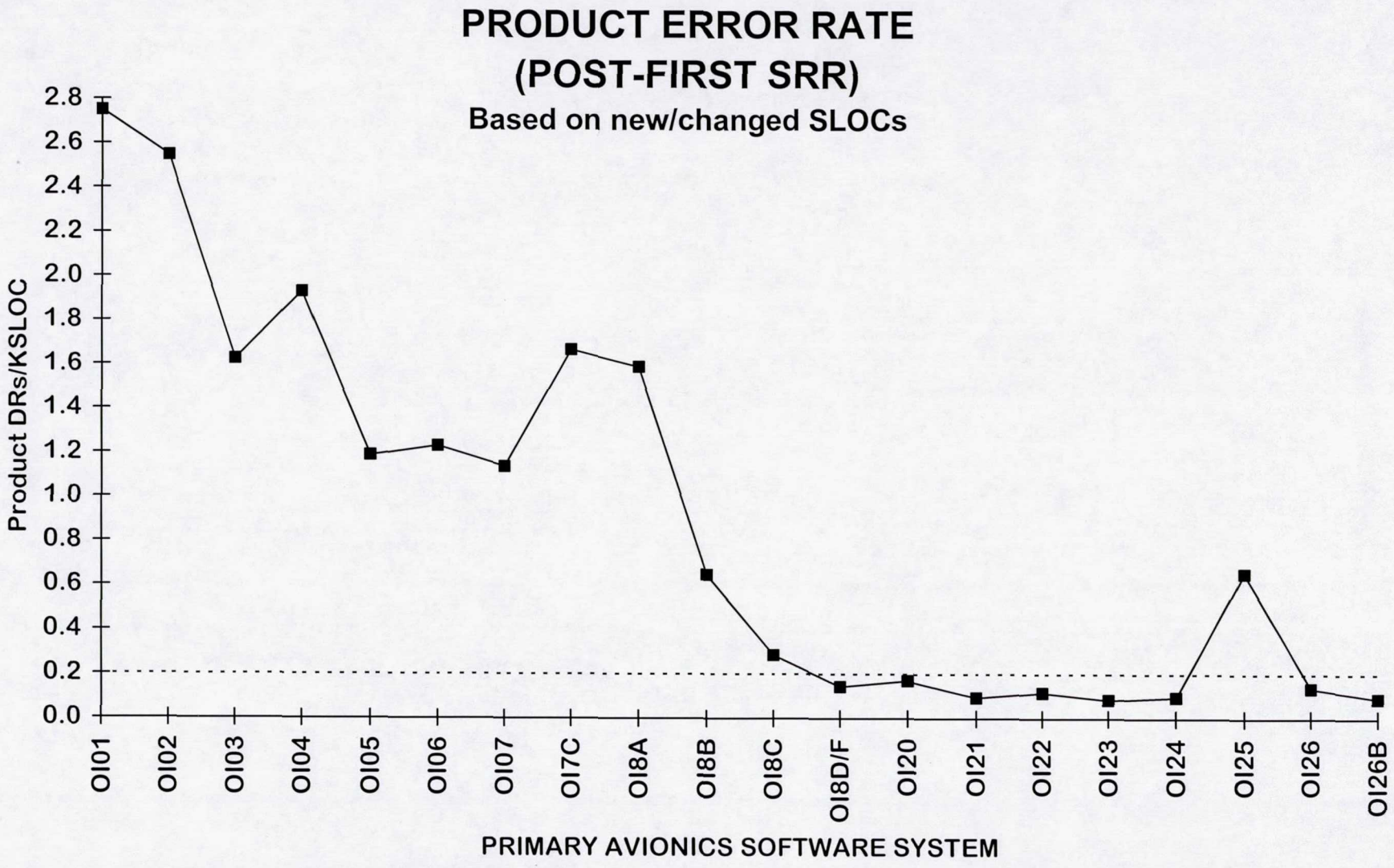
CI26B082.XLS

Product Error Rate

$$\frac{\text{\# of Valid DRs Post Completion}}{\text{New/Changed KSLOCs}} = \text{Product Error Rate}$$

- **The Product Error Rate measures the quality of the flight software after it has been released to the customer and certified for its first flight.**
- **In order to provide an indication of the quality of the flight software after release to the customer at CI, a variation of the Product Error Rate known as the *Released Product Error Rate* is also computed.**
  - The Released Product Error Rate includes all process DRs identified post-CI plus all product DRs.

FSW Product Error Rate



SW03BC.XLS

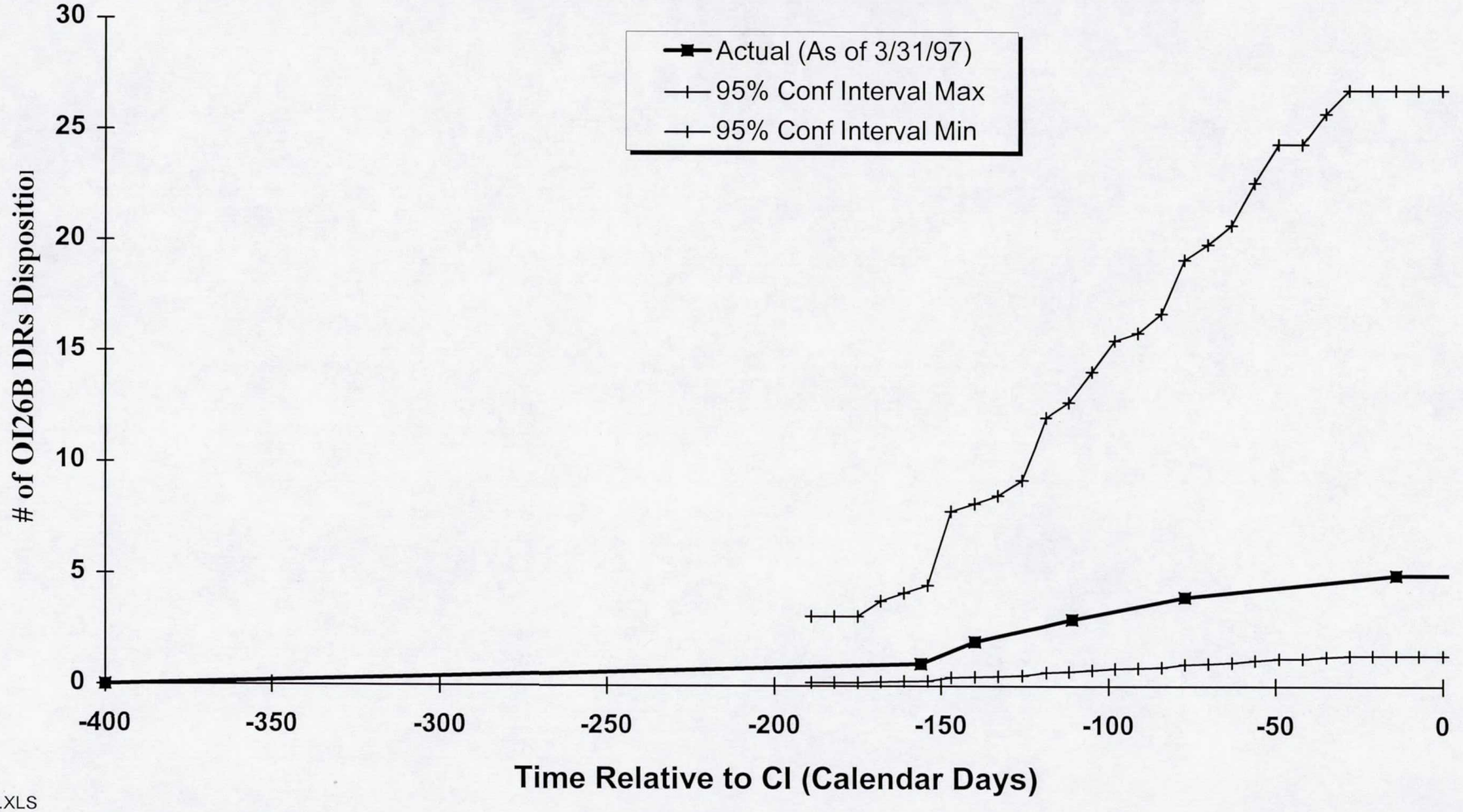
Analysis and Use of Data

- **Example types of analysis**
  - Comparison of predictions to actuals
  - Variance in confidence intervals
  - Progress over time in error detection rate compared to prior systems
- **Management sees the key project level metrics at least bi-weekly, if not more often, and are cognizant of the quality of each released system and of the systems with near-term releases**
  - Data generally within expected performance regions and infrequent signals
  - Direct actions when out of limits or out of predicted performance margins
- **Limited automated analysis tools in use**

Sample Error Detection Over Time Within a System

OI26B DR Dispositions Versus Time

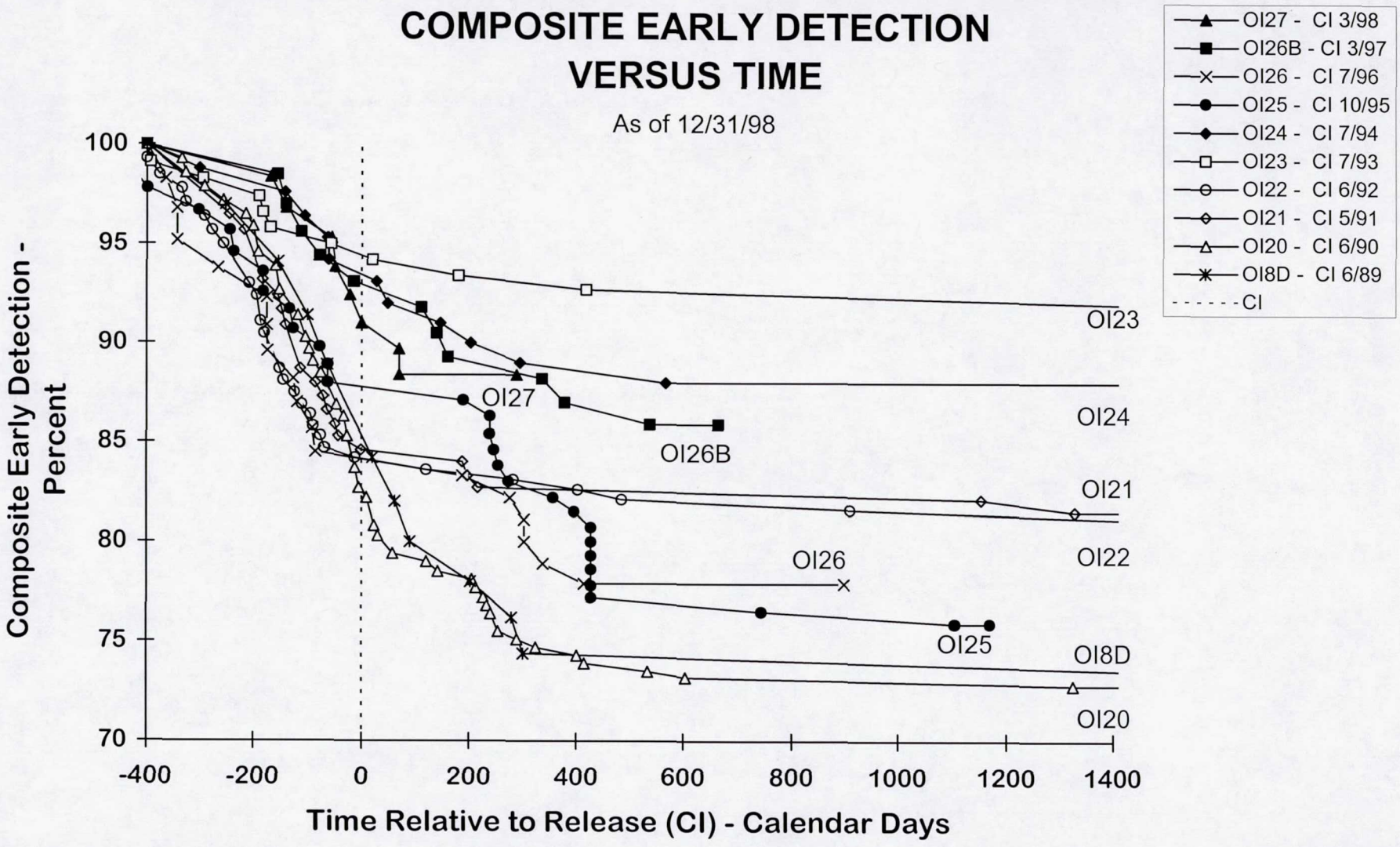
Using OI8D-OI26A Process Trend and 67 Major Errors at 26.41



DRVTM26B.XLS

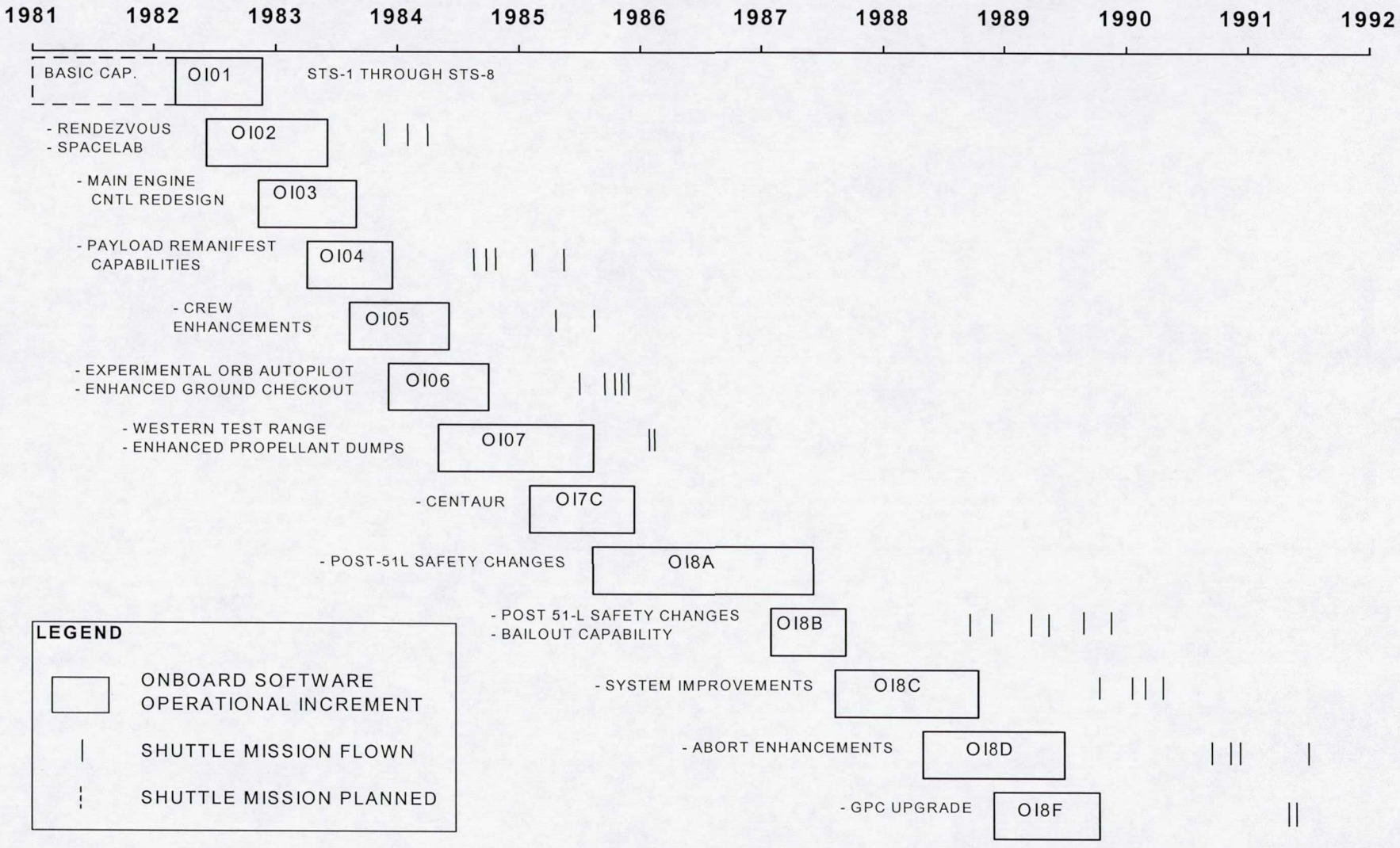
Sample Error Detection Over Time Across Systems

PRIMARY AVIONICS SOFTWARE SYSTEM



CI01.XLS

SOFTWARE RELEASE SCHEDULES



SOFTWARE RELEASE SCHEDULES