Evaluating Software Development Environments

Brendan Murphy

Microsoft Research Cambridge

Talk Overview

- History of Software Metrics
- Defining Clear Goals
- Review of Metrics
  - Contextual
  - Constraints
  - Progression
- Metrics Challenges
- Interpretations advice

History of Software Metrics

- Initial focus was computer manufacturers, influenced from hardware metrics
  - Data collected as part of service
- Jim Gray's 1980's paper highlighting the human impact on failures
- 1990's processes and metrics were in vogue (ISO 9000, SEI etc.), to protect against Y2k.
- Agile became in vogue, less process, less metrics

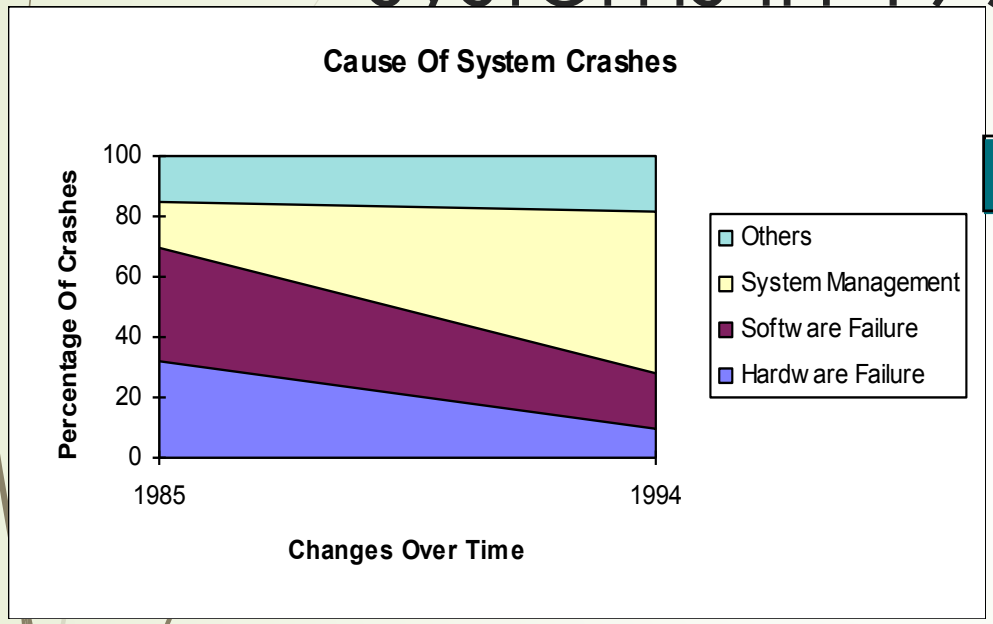
History of Metrics

- Models began emerging in the 60's being derived theoretically or empirically.
- Models initially focused on reliability and cost models became popular for a while.
- A significant proportion of papers at most SE conferences are about models, for many software attributes
- These models are often product version specific
  - Rarely replicated on other products or even other versions of the same product.

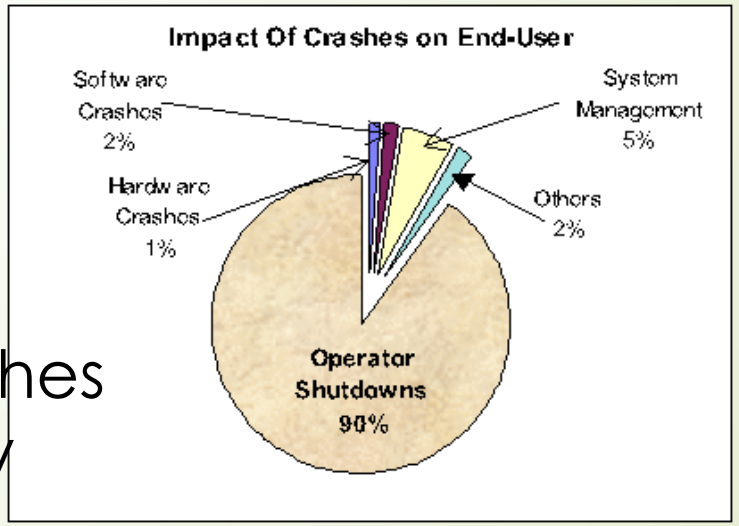
Why no universal model(s)?

- In engineering universal models assume
  - Common development processes
  - Common definition of failure
  - Driven by safety considerations
    - Often after the fact
- These are not universally applicable to software!
  - No clear definition of failure

Behavioural trends of VAX systems in 1990's



Impact on Availability?

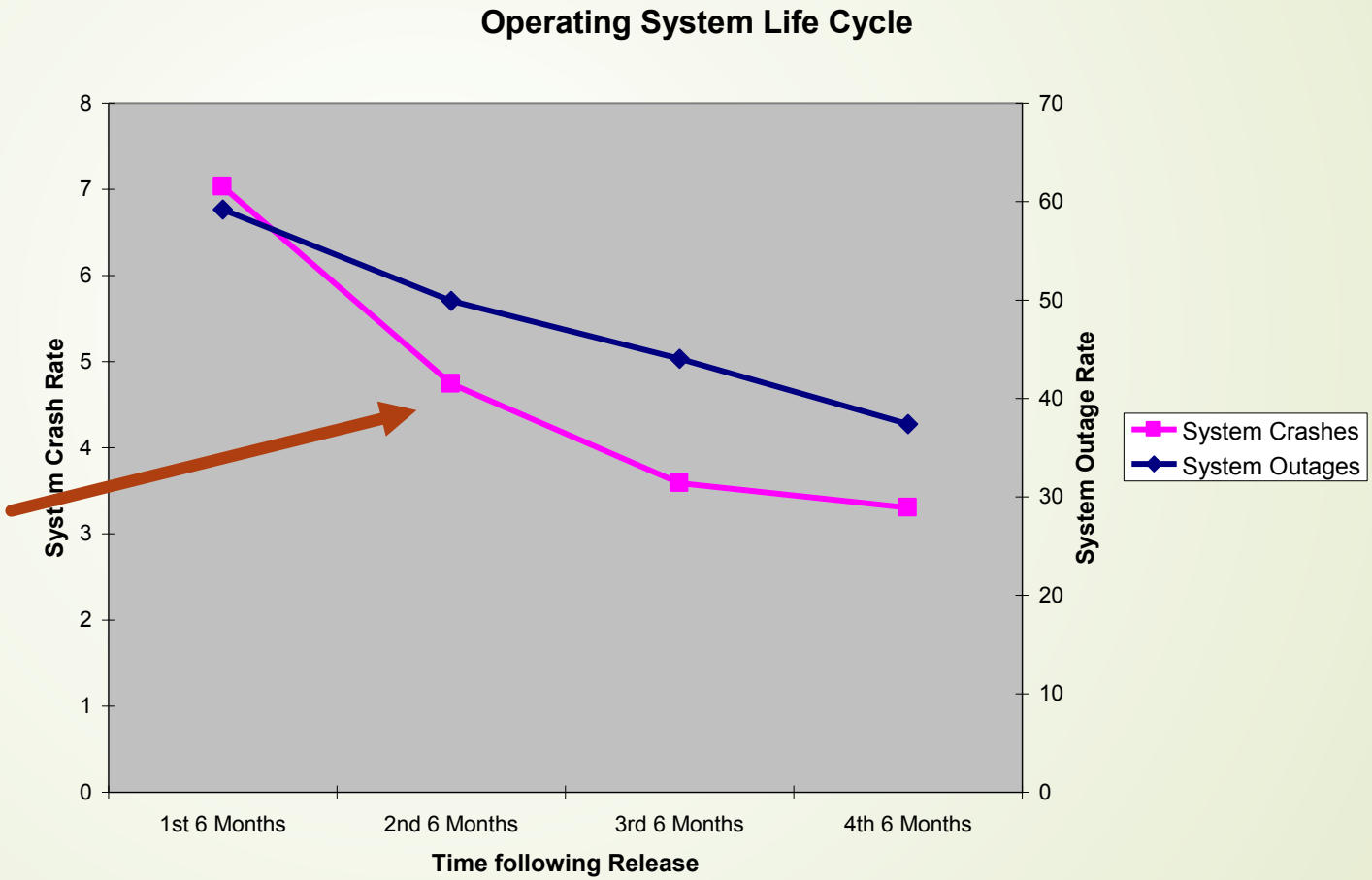


Customer less bothered about crashes
More bothered about unavailability

Murphy, Gent, Measuring System and Software Reliability Using an Automated Data Collection Process; Quality and Reliability Engineering Vol 11, 341-353

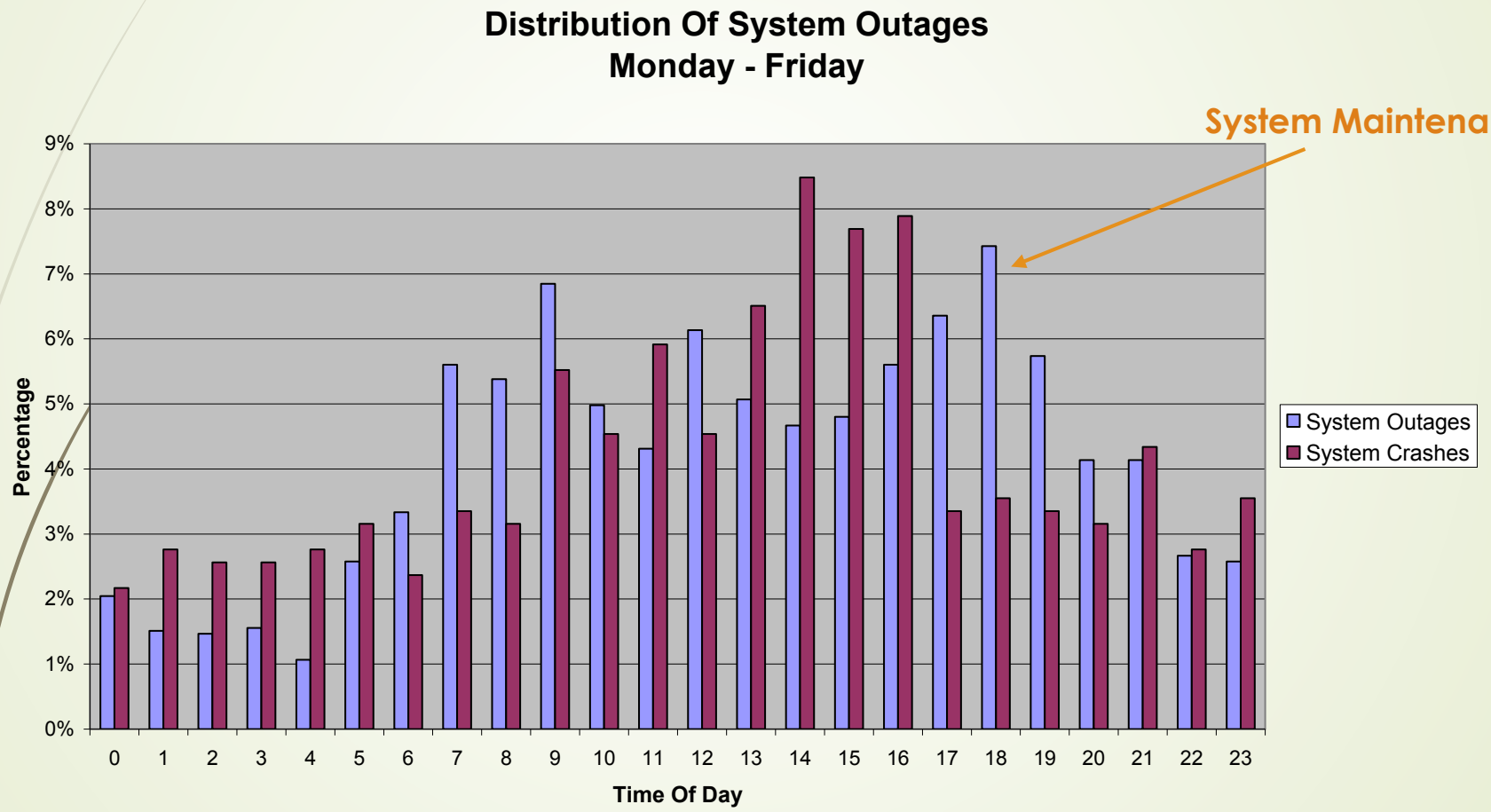
Not only a technology Problem VMS in the 90's

Trend not driven by software updates



Murphy, Gent, Measuring System and Software Reliability Using an Automated Data Collection Process; Quality and Reliability Engineering Vol 11, 341-353

The Human Problem



Brendan Murphy: Microsoft Research

Murphy, Davies, System Reliability and Availability Drivers; FTCS Conf Madison 1999

Software Variability

- Software attributes
  - Aircraft control software versus Phone game app
- Software methods
  - Agile versus waterfall
- Software Failures
  - Data corruptions versus UI failure
- Customer understand the relationship between new features and quality
  - Phone camera's

Underlying Assumptions

- Software diversity means unlikely to be a gold standard for
  - Metrics
  - Models
- No universal development process
  - Driven by team preferences and cost of failures
- Failures are product and context dependent

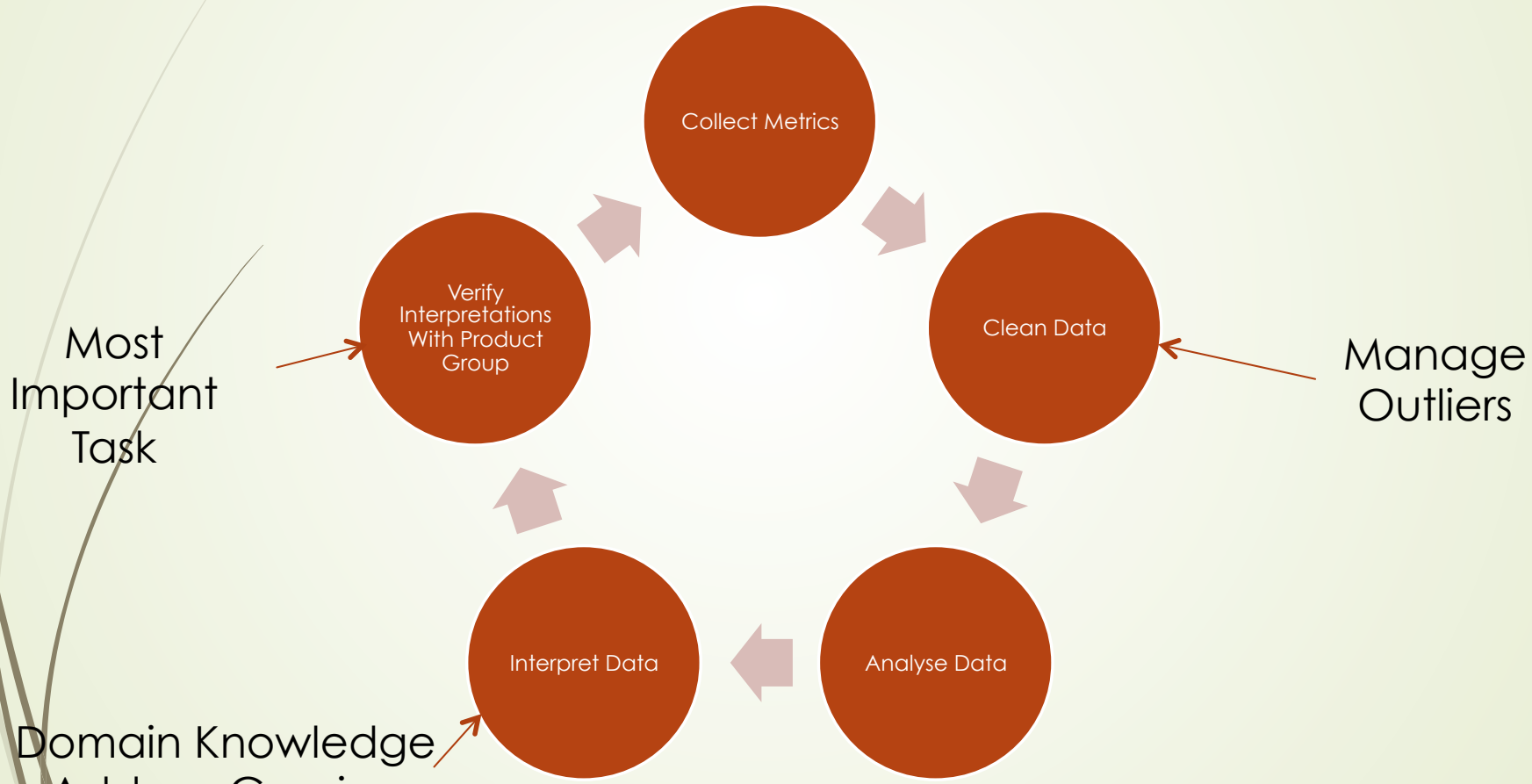
Establishing Clear Goals

- Goals must be
  - Relevant
  - Achievable
  - Measureable
- Based upon
  - Past product
  - Product goals
  - Version goals

Developing Goals through Benchmarking

- Main Benchmark Objectives
  - Characterizing the product or product family
  - Understand the relationship between metrics and reality
- Side Effects
  - Better understanding of the product
  - Understand the data collection process
  - Understand the 'cleanliness' of the metrics
  - Understand the amount of gaming in the system
  - Verify Interpretation

Benchmarking Process



Product Goals

~~Traditional Type Goal~~

~~Increase Quality, Productivity and Functionality by 30%~~

- Goals should be achievable based on the benchmarking of past products
- Goals should be relevant to the release
  - Is the next release a full feature release or a service release?
- Goals should be measureable
  - If they are not then a mechanism should be developed to ensure they are.

Metrics: Considerations

- Product Characteristics
  - Stand Alone product, Application deployed across multiple platforms, a service product?
  - Objective of specific release?
- Development Process
  - Impacts interpretation of metrics
- Release and Support Characteristics
  - Service or product
  - Cost of a Bug
  - Support for past products

Metrics: Characterization

- Contextual Metrics
  - Product and organizational domain Knowledge
- Constraint Metrics
  - Non Version specific metrics (security, power, storage etc.)
- Progression Metrics
  - Status of the development
  - Quality evolution
  - Developer efficiencies

Contextual Metrics Captured Manually

- Product Attributes
  - Objectives (business goals), Audience, delivery
- Project Attributes
  - Scope of version release, relationship with historical releases, areas of concern, technical debt
- Organizational Attributes
  - Relationship to Conway's law, decision making, interface between groups, retained knowledge
- Development Attributes
  - Development methodology (changes?), development tools

Constraint Metrics: Reliability

- Characterize total reliability of the product
  - Verify through 'Dog Fooding' or BETA releases
  - Services through releases to sub set of users
- Many predictive models, few are repeatable
- Future reliability should be relative to past product
  - Installation
  - Failure Rates
  - Availability
  - Recoverability
- Reliability Goals driven by release type
  - Large feature release will see a drop in reliability
    - Features rarely used as designed
    - Users tend to play around with new features
  - Service Pack Releases will see an improvement

Constraint Metrics: Performance

- Large impact on Customer Satisfaction
- Attributes
  - Resource consumption
  - Response Time
- Contributing Factors
  - Environments
- Metrics expressed as distribution over population
- Goals measured relative to past versions or competitors
  - Measured at the system level
  - Impact of component changes on system performance may be difficult to predict

Constraint Metrics:

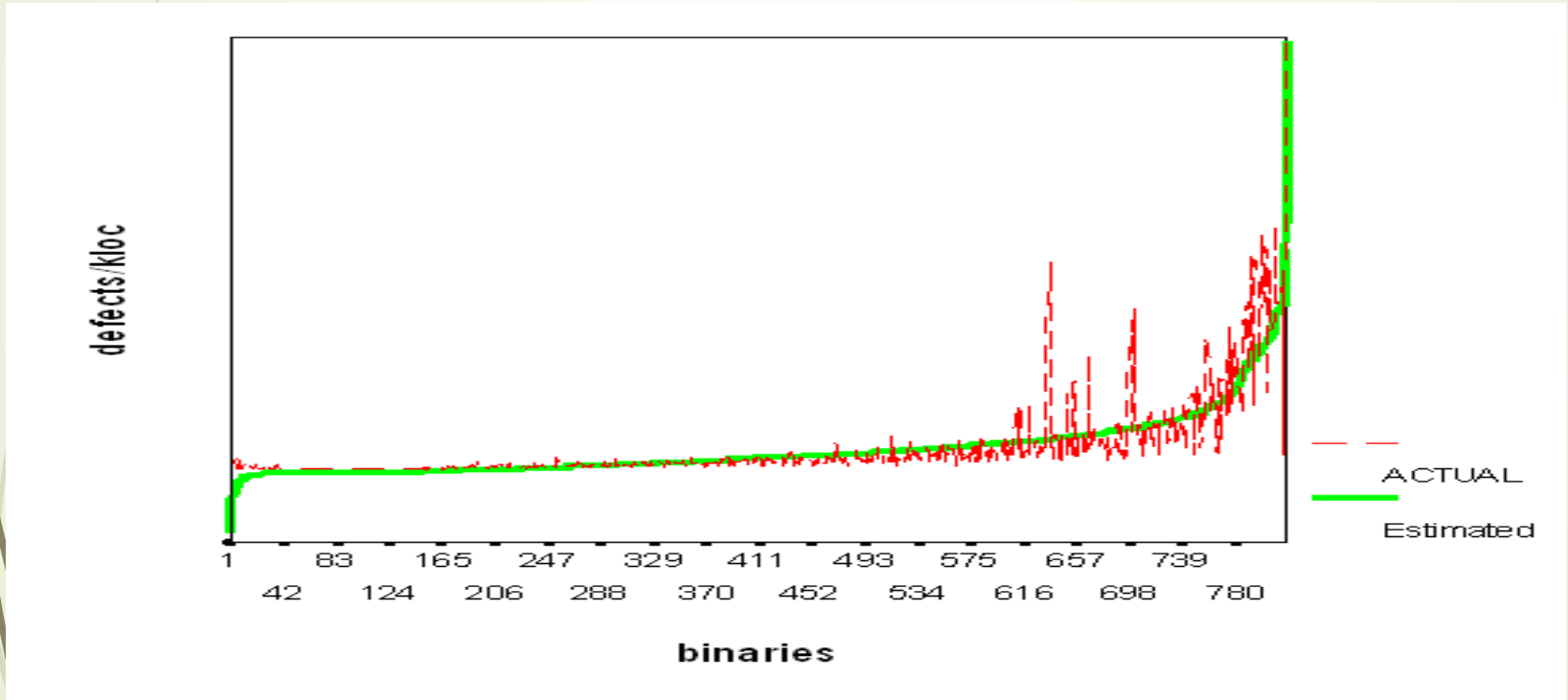
- Backward Compatibility
  - API
  - 3rd Party applications
- Security
  - Compliance with laws (privacy)
  - Threat Models
    - Create new attack vectors
  - Verification
    - SDL
    - Fuzzing
    - Static analysis…..

Progression Metrics: Churn

- The rate of change of code
  - File changed
  - Lines changed (added, deleted or modified)
- Characterize what is being monitored
  - The product
  - Configuration controls
  - Service Process scripts
  - Development Process Scripts
  - Test code
- Interpreted based on the project milestone

Progressions Metrics: Churn

- A good predictor of Failure



Use of Relative Code churn measures to predict System Defect Density, ICSE 2005
Nagappan, Ball (Microsoft)

Progression Metrics:

- Complexity
  - McCabes cyclomatic complexity
    - Effective measure of the testability of software
    - Many opinions on the effectiveness of the measure in terms of goodness
  - Important attribute is the changes complexity between releases
- Dependencies
  - Relationship between modules within Binaries
  - Relationship between binaries
  - Often used as a measure of complexity
  - Analysed to identify cyclic dependencies

Progression Metrics

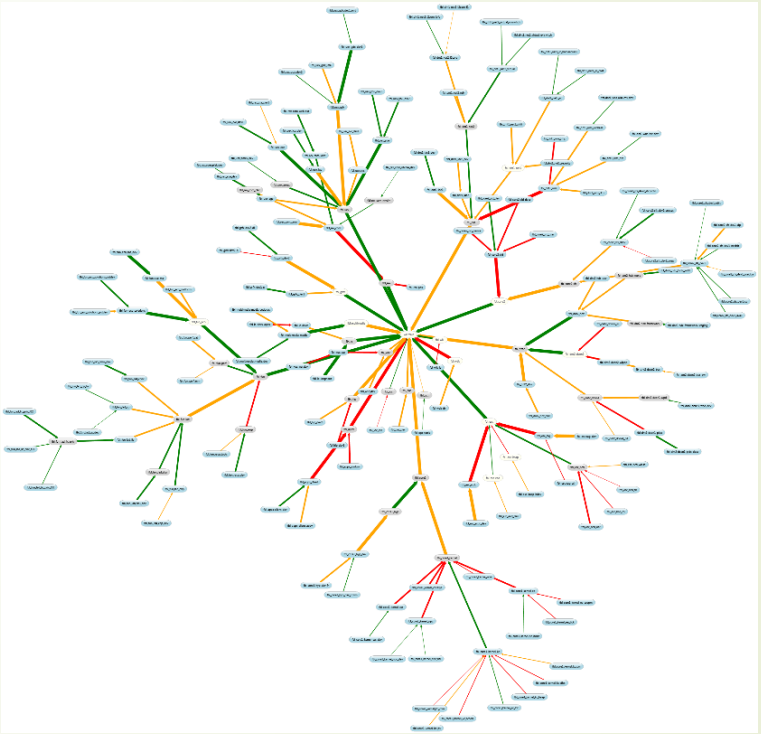
- People
  - Ownership
  - Knowledge
- Legacy
  - The amount of old code in the product
  - Old code attributes
    - Reliability
    - People are reluctant to change as knowledge is lost
    - Security and architectural risks.

Progression Metrics: Quality

- The most monitored metric
- Often the least accurate due to
  - Loose definition of failures
  - Metrics used to measure testers and developers and therefore often gamed
  - Rate of bugs are similar across multiple products (non quality related)
  - The less the amount of bug triage the less relevant the metric
- Metrics measured
  - Bugs found during a period of time
  - Bugs resolved rate
  - Bugs resulting in code changed (preferred measure)
  - Unresolved bugs

Code Velocity

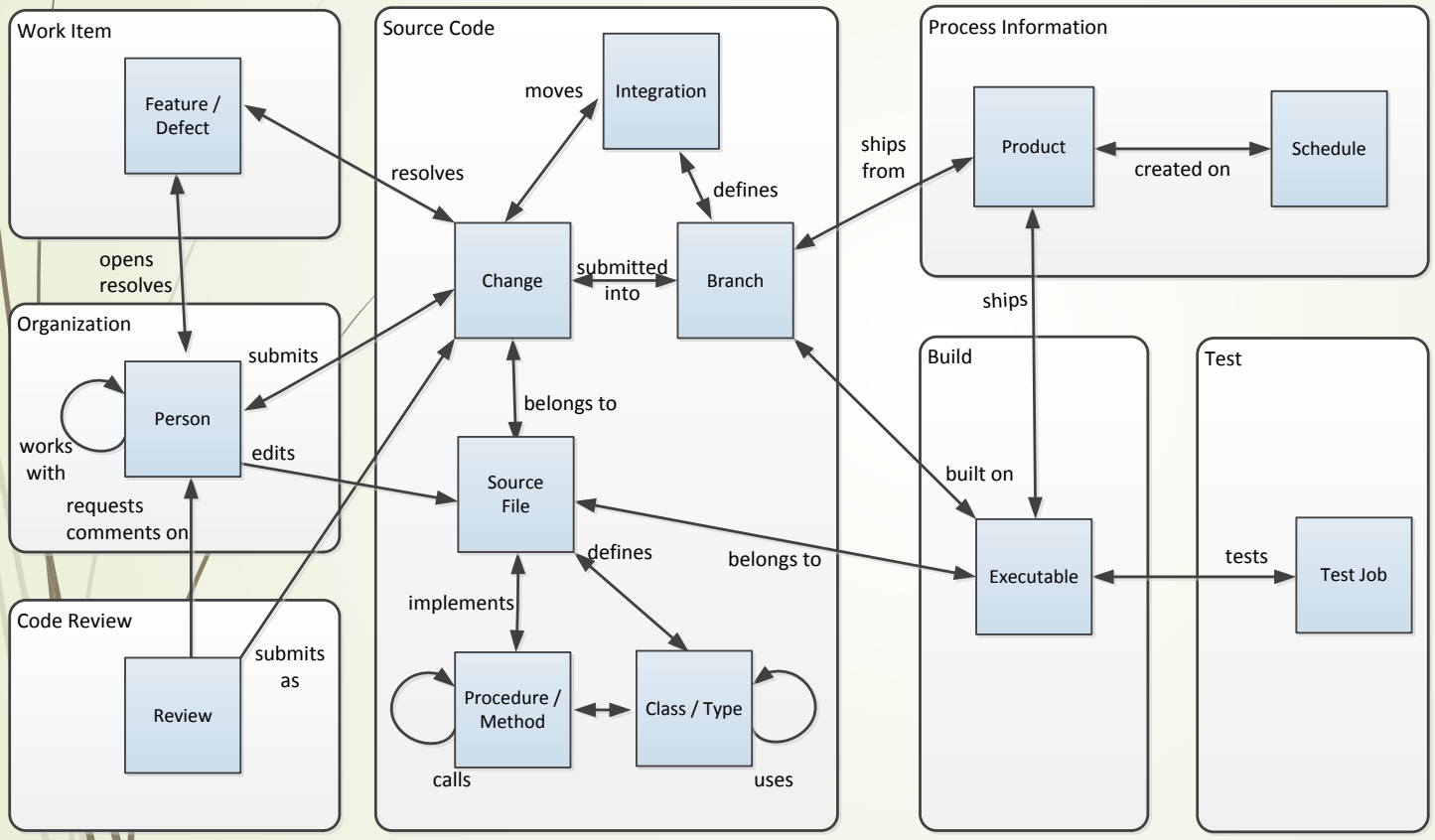
- The time for code to
  - Progress through the development process
  - Deployed
  - Feature complete
- Process efficiency
- Engineering productivity
- Team Productivity



Challenges to characterizing Software Development

- Data Collection: Methodology
  - Manual
    - Surveys, interviews, manual inspection
    - Simple but expensive and time consuming
    - Captures intent and perceptions
  - Automatic
    - Uses existing data sources like Code Management Systems.
    - Needs manual verification and noise reduction
    - Data easier to analyse more difficult to interpret
    - People often confuse data with facts.

Data Collection at Microsoft: CodeMine



http://research.microsoft.com/apps/pubs/default.aspx?id=180138

Why was data Collected?

- Ideal are data from tools designed to manage the workflow or process being managed
- Problematic is indirect data collection
  - Counting bug fixes in quality databases
  - Where interpretation occurs then it is dependent on the quality of the interpretation
- Bad metrics are those that have been used to measure people
  - Assume these are gamed.

Process Semantics

- Workflows are not explicit or documented
  - E.g. code flow through the CMS system
  - Can influence interpretation
- Bugs are the biggest issue
  - Source determines the quality of the bug
  - Customer bugs can be subjective (UI quality)
  - Changes to Triage process impacts metrics
  - Not all bugs are recorded

Common interpretation Issues

- Treating metrics as facts
- Noisy data
- Gaming
- Outliers

Interpretation Advice

- Focus on System metrics not just component metrics
- Monitoring Quality
  - Regular builds
  - Measure feature complete
  - Component stabilization
- Tracking Progress at high level
  - Ensure the churn metrics reflect product plans
- Verifying metric interpretations against perception

Summary

- No generic solution
- Establish Clear metrics
- Metrics
  - Contextual
  - Constraints
  - Progression
- Challenges
- Interpretation hints


Questions