Evaluating Empirical Studies

Dewayne E Perry
ENS 623
Office Hours: T/Th 11-12
322C – Spring 2005

Evaluation Outline

- Review two studies
  - Look at the design and results of each paper
  - evaluate according to our guidelines for good and credible studies
- Use our own work

Review Papers

- Software Fault Study
  - Perry and Steig, "Software Faults in Evolving a Large Real-Time System: A Case Study", ESEC93, Sept. 1993.
- Time Study
  - Bradac, et al., "Prototyping a Process Monitoring Experiment", IEEE TSE, Sept. 1994.
  - Perry, et al., "People, Organizations, and Process Improvement", IEEE Software, July 1994.

Experimental Site

- Large-scale, real-time software system
- C Programming language, with some domain specific languages
- UNIX development environment
- Feature is the unit of development
- All changes via Change Management System (CMS)

Software Faults - Research Context

- Error studies have usually been done in context of initial and not evolutionary development
- Interface errors studies of Perry/Evangelist showed the importance of interface problems in evolutionary development.

Software Faults - Research Question

- Were application specific faults the critical problems in a particularly faulty release?
- What classes of faults were there and when were they found?
- How hard were they to find and fix?
- What were their underlying causes?
- What means could be applied to either prevent or alleviate them?

Software Faults - Experimental Design

- Two phase study
  - Investigate the entire set of faults
  - Investigate the largest subset (design and implementation)
- Data capture from owners of faults when closed
  - Members of development part of team to design the survey
  - Development volunteers to review/pre-test the instruments
- Management imposed limitations:
  - Strictly voluntary participation
  - Complete anonymity of responses
  - Completely non-intrusive

Software Faults - Phase 1

- Problem categories:
  - previous, requirements, design, coding, testing environment, testing, duplicates, no problems, other
- Test phase when found:
  - capability test, system test, system stability test, alpha test, released

Software Faults - Phase 1 Results

- Response rate of 68%
- 34% development
  - ★ requirements (5%), design (11%) and coding (18%)
- 25% testing
  - ★ testing(6%) and environment (19%) problems
- 30% overhead
  - ★ duplicates (14%) and no problems (16%)
- 11% other

Software Faults - Phase 1 Summary

- Requirements, design and coding faults were found throughout all testing phases
- Majority of faults were found in system test and late in the testing process
- The evolution of large, complex software systems involves a large overhead

Software Faults - Phase 2

- Fault types (design and coding):
  - language pitfalls, protocol, low level logic, change management system complexity, internal functionality, external functionality, primitives misused, primitives unsupported, change coordination, interface complexity, design/code complexity, error handling, race conditions, performance, resource allocation, dynamic data design, dynamic data use, static data design, unknown interactions, unexpected dependencies, concurrent work, other

Software Faults - Phase 2

- **Cost information**
  - ★ Ease of finding or reproducing the fault
    - ➢ Easy - could reproduce at will
    - ➢ moderate - happened some of the time
    - ➢ Difficult - needed theories to figure out how to reproduce
    - ➢ Very difficult - exceedingly hard to reproduce
  - ★ Ease of Fixing the fault
    - ➢ Easy - less than a day
    - ➢ Moderate - 1-5 days
    - ➢ Difficult - 6-30 days
    - ➢ Very difficult - greater than 30 days

Software Faults - Phase 2

- Root cause and solution
  - Underlying causes:
    - none give, incomplete/omitted requirements, ambiguous requirements, incomplete/omitted design, ambiguous design, earlier incorrect fix, lack of knowledge, incorrect modification, submitted under duress, other
  - Means of prevention:
    - formal requirements, requirements/design templates, formal interface specifications, training, application walk-throughs, expert person/documentation, design/code currency, guideline enforcement, better test planning, other

Software Faults - Analyses

- Test for pair-wise independence
  - Chi-Square test:
    - if observed is the pairwise product, then the variables are independent
    - if observed is not the pairwise product, then they are not behaviorally independent
  - Example - using find and fix data (assume 1000 responses)

fix (e+m, d+vd)		784	216
find (e+m, d+vd)	909	713 (725)	196 (184)
	91	71 (59)	20 (32)

  - None of the relationships were independent
    - means of prevention and ease of finding had least significant dependence
    - root causes and means of prevention had most significant dependence

Software Faults - Analyses

- On the basis of the Chi-Square test, we concluded the following were correlated:
  - ★ costs and faults
  - ★ costs and underlying causes
  - ★ costs and means of prevention
  - ★ underlying causes and means of prevention
  - ★ interface and implementation faults

Software Faults - Results

- Response rate of 68%
- The variables were not independent of each other
- Lack of information tended to dominate the underlying causes
- Knowledge intensive activities tended to dominate the means of prevention
- Informal means of prevention were preferred over formal means
- Interface faults were harder to fix than implementation faults

Software Faults - Evaluation

- Better empirical studies
  - ★ Answers an important question
    - ➢ Yes: What are the significant development problems
  - ★ Establishes principles
    - ➢ Yes: Knowledge issues are fundamental problems
  - ★ Enables generating and refining hypotheses
    - ➢ Exposes a number of interesting problems
  - ★ Cost effective
    - ➢ Inexpensive design/implementation
    - ➢ Expensive analysis (people intensive)
  - ★ Repeatable
    - ➢ useful design; expect similar correlations, not same results

Software Faults - Evaluation

- Credible interpretations –
  - ★ Strengths in construct, internal and external validity
    - CV: Important variables
    - IV: Instrument created by developers themselves
    - IV: Random trial with developers
    - IV: Data from people who owned the fault solutions
    - EV: Release similar to other releases
    - EV: Commonly used language and environment
    - EV: Response rate of 68%
  - ★ Limits/Weaknesses in construct, internal and external validity
    - CV: Find, Fix interpretation not identical
    - CV: Fault categories poorly structured (too many faults, etc)
    - IV: No post survey validation - only pre-survey
    - IV: Up to a year lapse between problem resolution and survey
    - IV: Analysis weakened by find/fix problem
    - IV: Interface/Implementation division not clean
    - IV: Effect of 32% not returned
    - EV: Single case study, single system
    - EV: Single domain

Software Faults - Evaluation

- Credible interpretations - continued
  - ★ Test hypotheses
    - ➢ Yes - refuted the hypothesis that application specific faults were the critical faults
  - ★ Adequate precision
    - ➢ Over two thirds results - significant set of responses
    - ➢ Three place precision is justified by the response volume
    - ➢ dependence/independence analysis
    - ➢ correlations of fault factors
    - ➢ comparison of interface and implementation faults
  - ★ Available to public
    - ➢ Lack of absolute numbers
    - ➢ Basic data is not provided in paper, only summaries of analysis

Software Faults - Summary

- Useful case study - answers important questions
- Done within limitations of management constraints
- Significant effect on internal development process
- Important for research implications
- Weaknesses in the survey instrument
- Questions about generalizability

Time Studies

- Three Studies (Iterations)
  - ★ Longitudinal study of a single developer, single development (Prototype ...)
  - ★ Self-reporting study of multiple developers/developments (People ...)
  - ★ Direct observation of a subset of those developers (People ...)

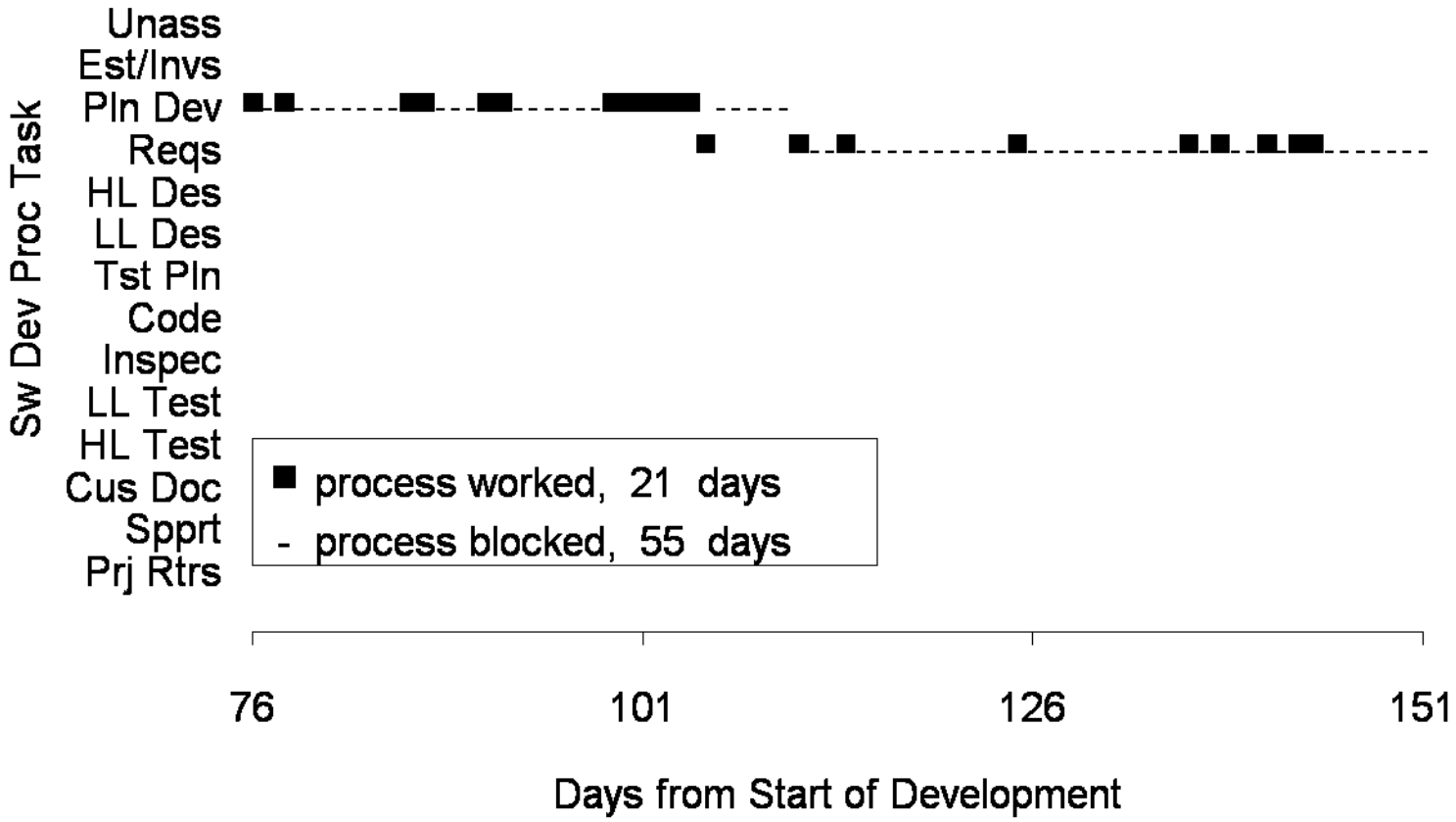
Time Studies

- Research Context
  - Single programmer studies usually in context of simple problems
  - Few studies of programmers in the context of team
  - Few studies of programmers in the context of teams in large-scale software development
- Research Question (Hypothesis)
  - How does a developer spend his or her time in the context of a team development as part of a large system development?
  - What effects do inter-team/personal dependencies have?
  - How much time is spent in communication?
  - How much time is spent in the relevant processes? Where?
  - How much time is lost for various reasons?

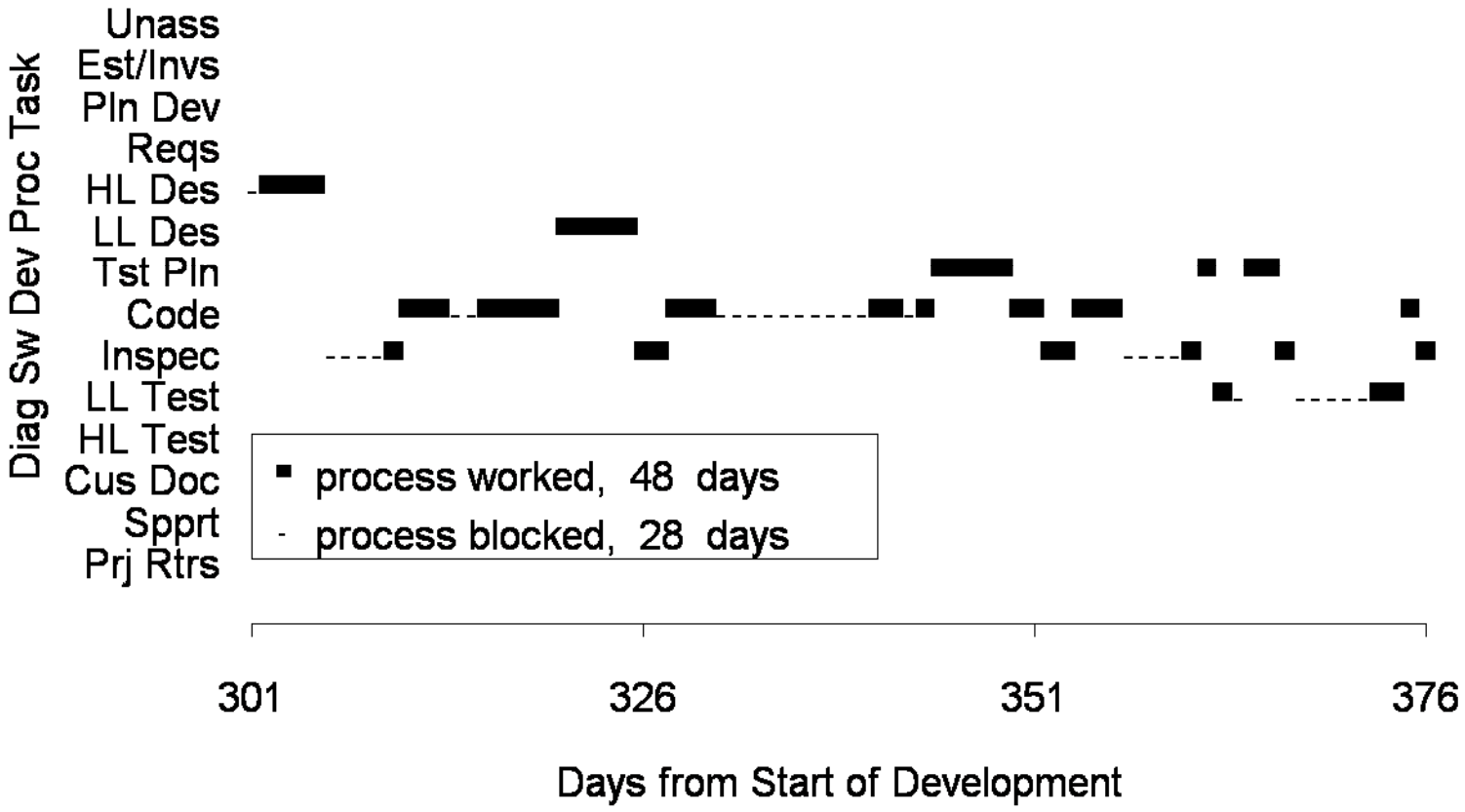
Time Studies - Phase 1

- Specific null hypothesis:
  - A person is 100% effective (ie, race time = lapse time) in the context of teams in large scale software development
- Experimental Design
  - Longitudinal study
  - Retrospective reconstruction of 32 month development from project notebooks and personal diaries.
  - Categorized time spent in the specific process activity:
    - working, documentation, rework, reworking documentation
  - Categorized how time was spent when not in process:
    - waiting on lab, expert, review, hardware, software, documentation, other

Time Studies - Phase 1 DataTime Spent Early



Time Studies - Phase 1 DataTime Spent Later



Time Studies - Phase 1 Results

- Race time / lapse time = .4
- Blocking significant
  - long significant periods early in the process
  - short periods in the middle - least blocking here
  - short periods, large amounts of blocking late in the process
- Process phenomenology
  - waterfallish early
  - iterative later
- Provides an important basis for iteration to delve deeper into the question of how developers spend their time.

Time Studies - Phase 2

- Research Context
  - Refines phase 1
  - Vertical slice of multiple developers and developments
- Research Questions (in addition to initial questions)
  - How significant was the Phase 1 study and where does its significance lie?
  - How representative was the subject used in longitudinal study?
  - Is blocking as significant a factor as in the initial study?
- Experimental Design
  - Self-reporting instrument - finer resolution
  - Activity and state of work for each process step in half/hours

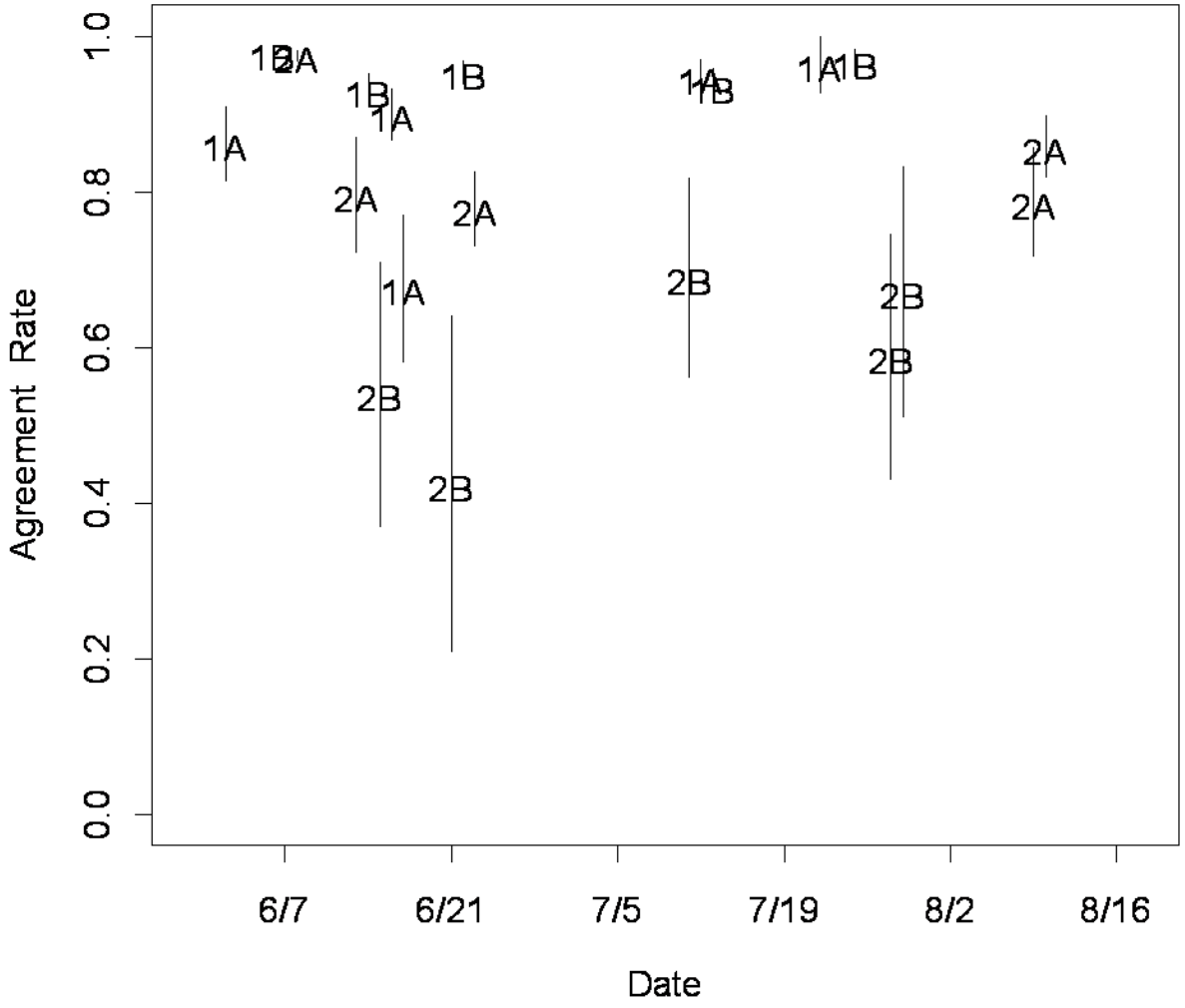
Time Studies - Phase 2 Results

- Confirmed race time / lapse time = .4
- Longitudinal study congruent with self-reporting study
- Blocked = context switching
- Clarifies our understanding of how developers spend their time
- Raises questions about variance of self-reporting

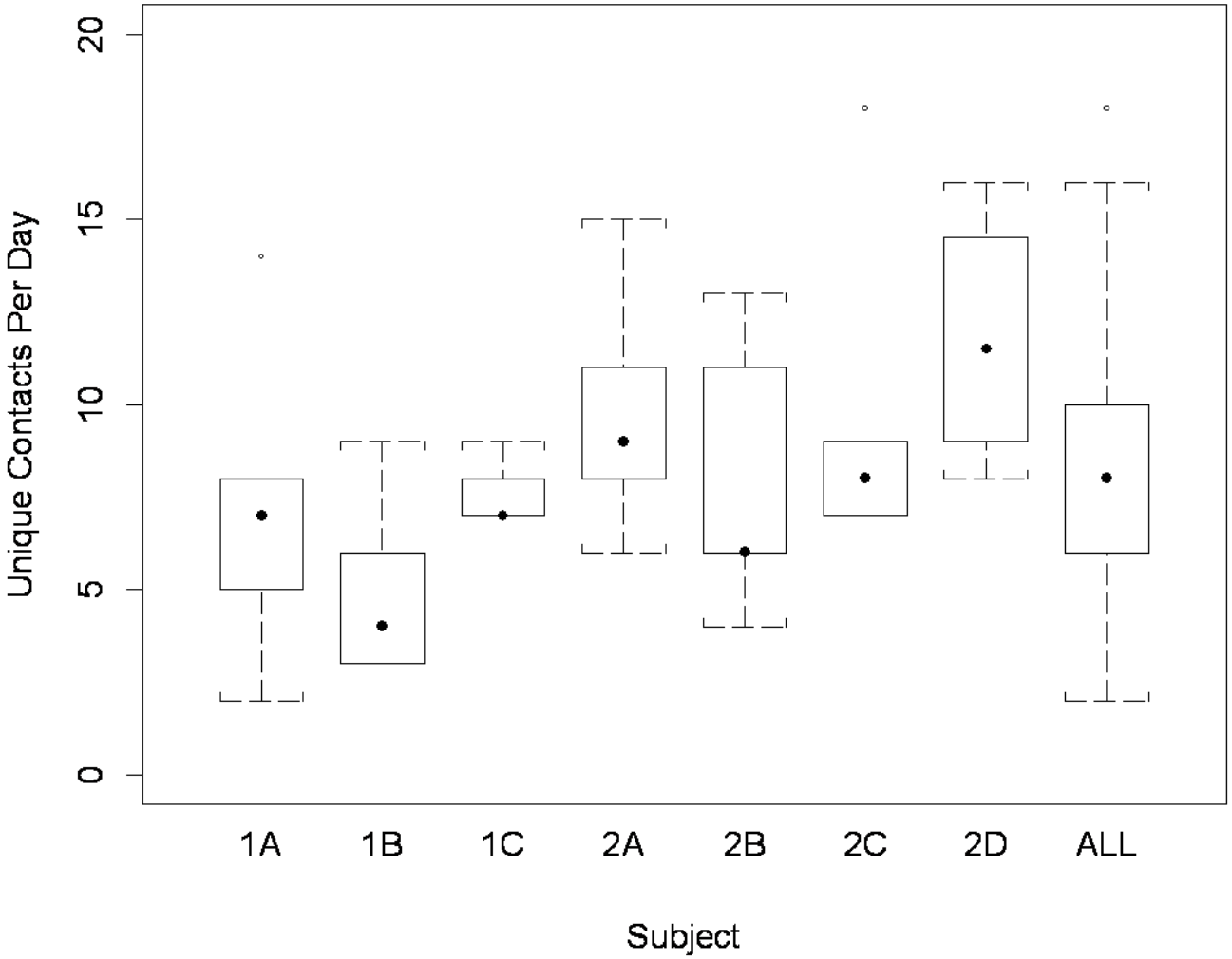
Time Studies - Phase 3

- Research Context
  - Self-reporting follow-on
  - A more detailed look at what developers do with their time
- Research Questions (Hypothesis)
  - How valid was self-reporting
    - What are the variances in self-reporting?
    - How close is the correspondence between perception and reality
  - What is there that happens at a finer time resolution than 1/2 hour?
- Experimental Design
  - Series of arranged full-day observations
  - Comparison of the observations with the self-reports

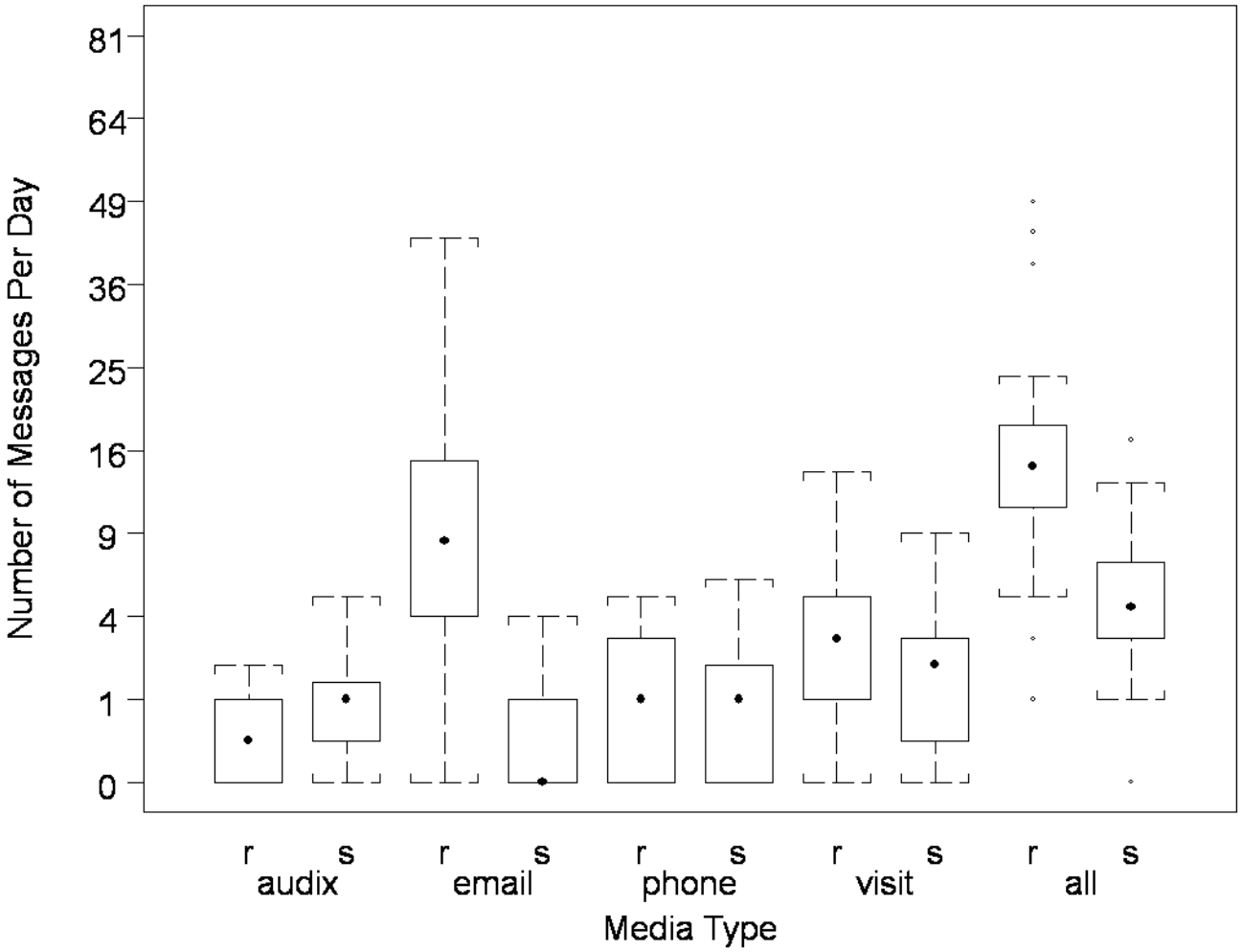
Time Studies: Phase 3 - Self-Report Fidelity



Time Studies: Phase 3-Unique Contacts PerDay



Time Studies: Phase 3-Nr of Msgs Per Day



Time Studies - Phase 3 Results

- Delineates reliability of self-reporting
  - ★ Self consistent but not uniform
  - ★ 20% variance between observed and report
- Clarifies further our understanding of the how developers spend their time
  - ★ Significant amount of unplanned interruptions
  - ★ 75 minutes average per day in informal communication
  - ★ importance of oral communication, avoidance of written
- Importance of informal communications in development processes

Time Studies - Evaluation

- Better empirical studies
  - ★ Answers an important question
    - ➢ Yes: how developers spend their time
  - ★ Establishes principles
    - ➢ Yes: race/lapse time, informal interactions
  - ★ Enables generating and refining hypotheses
    - ➢ Exposes a number of interesting problems
  - ★ Cost effective
    - ➢ Varying costs - dependent on resolution desired
    - ➢ Effective for the results desired
  - ★ Repeatable
    - ➢ useful design; expect similar correlations, not same results

Time Studies - Evaluation

- **Credible Interpretations**
  - ★ **Strengths in construct, internal and external validity**
    - CV: Complete data source over complete development
    - CV: Well-defined retrospective, self-reporting and observational structures
    - CV: Established process vs state in process
    - IV: Congruency of results
    - IV: Established self-report consistency and range of variance
    - IV: Varying degrees of resolution
    - EV: People in team context in large-scale software development
    - EV: Entire life-cycle
    - EV: Common language and development environment

Time Studies - Evaluation

- Credible Interpretations – continued
  - Limits/Weaknesses
    - CV: Blocked, context switching ambiguity
    - IV: Loss of details due to time passed
    - IV: Inaccuracy of self-reporting
    - IV: Observations effects
    - EV: Representativeness of application domain
    - EV: Cultural representativeness
  - Test hypotheses
    - Yes - refuted the hypothesis
  - Removal of alternative explanations
    - Exposed where critical problems were
  - Adequate precision
    - Differing degrees of resolution as needed
  - Available to public
    - Data in various useful forms or presentation

Time Studies - Summary

- Race time / elapse time = .4
- Blocking / context switching significant
- Developers consistent, but not uniform, in self-reporting
- Significant number of, and time spent in, informal interactions