Collaborative Software Design & Development

*Collaboration*

Dewayne E Perry
ENS 623A
Office Hours: T/Th 10:00-11:00
perry @ ece.utexas.edu
www.ece.utexas.edu/~perry/education/382V-s08/
Environment Support

Model of software development environments

SDE model = \{ policies, mechanisms, structures \}

Policies

Requirements imposed on the user of an environment

- Eg, static linker/loaders require all externally referenced names to be resolved before successful linking

Need not be hardwired

- Eg, in Darwin (Minsky’s law-governed system), policies are defined by declarative rules
- Eg, Osterweil’s process programming enables explicit programming of the desired policies

Supported vs enforced policies

- Supported: provides means of satisfying the policy
- Directly enforced: no other way to do it
- Indirectly enforced: environment supported, management enforced
Environment Support

 Structures

- Objects or object aggregates that mechanisms operate
  - Eg, files in Unix - basic ubiquitous structure
  - Eg, abstract syntax trees in integrated environments
    ✓ Problem - difficulty integrating new tools if use different structure
    ✓ Partial solution: Garlan’s mechanism for generating a common underlying structure for integrated tools

- The more sophisticated and plentiful the objects,
  - The more comprehensive the mechanisms
  - The richer the possible set of policies
  - Eg, an object base (and extension of a database)
  - Eg, Inscape’s semantic interconnection structure

- Structures impose limitations on the kinds of policies
  - Eg, Infuse enables richer policies regarding interaction and coordination but do not allow policy definition
Environment Support

Mechanisms

- The languages, tools and tool fragments
- Implement with structures the supported and enforced policies
- Some visible, some hidden
  - Unix – all available
  - Smile – hidden beneath a facade that provides higher level mechanisms
- Policies can be encoded explicitly or implicitly
  - Explicit – Darwin and Marvel’s rules
  - Implicit – most SDEs are implicitly coded
Environment Support

4 classes of SDEs
- Individual
- Family
- City
- State

Individual:

- \( \text{tool-induced policies}, \) 
- \( \text{implementation tools}, \) 
- \( \text{single structure} \)

Primary issue: construction
Mechanisms dominate
Examples:
- Tool-kit environments (eg, Unix)
- Interpretive environments (eg, Interlisp)
- Language-oriented environments (eg, Cornell synthesizer)
- Transformational environments (eg, Refine)
Environment Support

Family:

(,…, coordination policies),
(,…, coordination mechanisms),
(,…, special-purpose databases)

Primary issue: coordination

- Generally laissez faire - ie, loose coordination
  - On a farm, can do pretty much want you want with respect to vehicles and driving rules

Structures dominate

Examples

- Extended toolkit environments (eg, Unix with SCCS)
- Integrated environments (eg, Gandalf Prototype)
- Distributed environments (eg, DSEE [now ClearCase])
- Project management environments
Environment Support

City:

(..., cooperation policies),
(..., cooperation mechanisms),
(..., structures for cooperation)

Primary issue: cooperation

- Often much more enforcement involved

Policies dominate

Examples

- Infuse - enforced cooperation
- ISTAR - contract model of cooperation
Environment Support

State:

(,..., commonality policies),
   (... , supporting mechanisms),
   (... , supporting structures)
)

- Primary issue: commonality
  - Enables cross project movement
  - Reduces new project training

- Higher-order policies dominate

- Examples
  - Product line environments
Environment Support

Contributions of the model

- Distinguishes those aspects of an environment useful in evaluating SDEs
- The taxonomy delineates 4 important classes of interactions
- Individual and family models are [still] the current state of the art
- City models provide qualitative differences
- Proposed state model - starting to see examples here
Tool Supported Collaboration

- Codified by Fagan - 1976 - peer review of code
  - Fresh look with no/few assumptions
  - Less expensive to detect fault when inserted
  - Versus cost of testing, isolating, repair and retest

- Code inspection process
  - Generate inspection package
  - Individual preparation
  - Team analysis - fault collection
  - Repair
  - Follow up

- Factors in code inspections
  - Structures - how steps are organized into a process
  - Techniques - how each step is carried out
  - Inputs - reviewer ability, code quality
  - Context - interactions, project schedule, personal calendars
  - Technology - tool support
Inspection Considerations

Code inspections in geographically separated projects
- Time-zone mismatches
- Travel for inspection meetings – expensive
- Long distance mailing of inspection packages

These factors affect project interval

Possible solutions
- Teleconferencing
  - 2 hours on the speakerphone: mind-numbing
- Video conferencing
- Groupware

Unknown factors
- Affect on effectiveness
- Underlying costs
Technology Pull instead of Push

Our tool development approach

- Select improvement criteria
  - Often reducing cost and increasing quality
  - Ours: reducing interval
- Model the cost-benefit drivers of the process
  - Need theories that are
    - General
    - Causal
    - Suggestive of control strategies
      - Theories $\rightarrow$ factors $\rightarrow$ manipulation
- Understand current process and identify key problems
  - Enables prioritization
- Explore and evaluate alternative improvements
  - Empirical studies key
  - Key issues, risks and costs
- Build and evaluate preferred improvement
Potential Drivers

Process structure
- Team size key factor
  - Coordination of multiple teams requires time
  - Some changes had adverse effect on interval
  - Overall, little effect on interval

Process inputs
- Code size explained 50% of variation
  - But little effect on interval

Techniques – with, without meetings
- Traditionally more faults found in meetings
- Experiment: meetingless more effective
  - When effort for meeting applied to individual analysis, more defects found
Potential Drivers

- Techniques - defect detection methods
  - Scenarios:
    - higher detection rate
    - more effective for what scenario covered
    - No less effective
  - Checklist no more effective than ad hoc

- Process environment
  - Influences choices of developers
  - Hence, does influence interval
    - Tasks have priorities
    - Under heavy load, low priority tasks deferred
    - Lengthened interval
  - Approaching deadlines affects work priorities
Key Problems

Process Overhead

- Creating and distributing inspection documents
  - Particularly a problem in geographically distributed projects
- Inspection data used by other processes
  - Defects must be collected, processed and managed
    - Eg change management - to ensure all defects fixed
    - Entered, fixed, closed and signed off

Blocking due to synchronization and sequencing

- Blocking can lengthen interval
- General problem
  - Many tasks sequentially ordered
  - Some tasks must be synchronized
    - Eg, group meetings
  - Inspection tasks sometimes coordinated
Solution Space

Possible ways of reducing interval
- Eliminate paper generation
- Eliminate inspection meeting
- Share preparation results
- Overlap preparation and repair

Justification
- Opportunistic study
  - Desk vs meeting inspections
  - No difference in average fault density
- Argument for sharing results
  - Sharing would be no worse than keeping them private
  - Useful process gains in sharing
- Overlapping inspection and repair
  - Problems: premature repair and consistency
  - Given risks, decided not to do this
Traditional Inspection Process

MR list

sinspect

inspection package

printing tool

Distribute paper,
have a meeting

Web-based Inspection

HTML inspection package

WebServer

inspector annotations

Web Browser

Web Browser
if (CP_cdata_ptr->op_sem_e & CPSCH_C_SLOTTED) {

/* In slotted mode, but have to ensure that */
/* store a slot cycle index value greater than */
/* allowed by the system. The value is using */
/* algorithm to determine the SELECTED slot cycle */
/* moreover, it is sending up to us the preferred */
/* index which can be anywhere from 0-7 while we */
/* values 0 and 1 currently in our system */
 */
Experience and Evaluation

✦ Initial acceptance was excellent
  ◆ Cost savings
  ◆ hypeCode integrated seamlessly into the existing environment
  ◆ New possibilities for concurrency and inherent speedups
  ◆ Web a natural and ubiquitous platform

✦ Anecdotal evidence interviewing users
  ◆ 10-25% reduction in total inspection interval
  ◆ Eliminates time spent in reproduction and distribution
  ◆ Improves effectiveness and reduces time spent
    ✓ Can see others comments
    ✓ Time spent detecting same defects eliminated
Experience and Evaluation

- Primary work product of meetings a by-product here
  - Eliminates about a day needed to prepare the meeting report
- Inspection report improved
  - More low, medium and observation category defects
  - No fewer severe category defects
  - Defect descriptions more detailed and complete
  - Recorded discussions of defects
- Need for collection meeting often eliminated
  - Reduces interval eliminating delay for meeting
  - When there is a meeting, very focused
  - Quality often higher – more focused on a few unresolved issues
  - Meeting significantly shorter, and less lapse time to meeting
    - Average about a half hour

- Overall a run away success – lost control of it
- Used on at least a dozen projects in half a dozen countries