Process Systems

- Process is both a technical and a managerial problem
  - Technical importance
    - Defined processes enable us to move from a *craft* to an *engineering* discipline
      - ✓From secrets passed from master to apprentice
      - ✓To published processes that can be
        - Scrutinized
        - Compared
        - Evaluated
    - Guide and coordinate very dynamic and complex processes
  - Managerial importance
    - Well understood practices are easier to manage
    - Common processes across projects
      - ✓Enables reallocation of people as resources
    - Greater predictability and tracking

Capability Maturity Model (CMM)

- Level 1 – Chaotic
- Level 2 – Repeatable
  - Requirements management
  - Software project planning
  - Software project tracking and oversight
  - Software subcontract management
  - Software quality assurance
  - Software configuration management
- Level 3 – Defined
  - Organization process focus
  - Organization process definition
  - Training program
  - Integrated software management
  - Software product engineering
  - Inter-group coordination
  - Peer reviews

Capability Maturity Model (CMM

- Level 4 – Managed
  - Software quality management
  - Quantitative process management
- Level 5 – Optimizing
  - Process change management
  - Technology change management
  - Defect prevention

Processes are Software Too!

- ICSE 1987 – Keynote Talk
  - Noted the similarities between processes and products
  - Introduced the notion of "process programming"
  - Given ICSE Most Influential Paper award at ICSE 1997
- A Software Process System goes through the exact same processes as a software product
  - The same life cycle
  - The same integral activities
- At detail level:
  - Disagreement about representation of processes themselves
  - Different kinds of measurement and evaluation

Processes are Software Too!

- Process models or process programs?
  - Models
    - Incomplete
    - Often weak on semantics, free for interpretation by people
    - Lack specifics that can be very important
  - Programs
    - Precise prescription for coordination of efforts of humans, computers and software tools
    - Executable
    - Blurs distinction between process and product
- Fundamental distinction
  - Who is in charge?
    - Human as a subroutine, directed by process support system
    - Human enacting process with HELP of process support system
  - People have different/varying process characteristics
    - Good at planning, handling exceptions
    - "machine language" level varies
  - Informal versus formal

Process Capture

- Analogous to requirements elicitation
- Three distinctions about processes
  - Process as defined
  - Process as done
  - Process as ought to be
- Useful mechanism for what is done: events
  - Event name
  - Time of occurrence
  - Who
  - Trigger
  - Response
- Best practices
  - Often based on experience, anecdotal evidence, some theory
  - Very little empirically validated

Process Architecture & Design

- Some Principles --- Generic Processes
  - Define process fragments
    - Combinable components
    - In terms of goals
  - Use appropriate means of abstraction (or generalization)
    - Parameterization
    - Primitivation – requires elaboration
    - Stratification - layering
  - Align activities with their appropriate processes
    - Eg, estimation is a project management activity, not a design activity
  - Separate project structure from process structure
    - project milestones and schedules,
    - project roles, obligations and permissions, and
    - the project's organizational structure.
- Some Principles --- Process Architecture
  - Modularize processes
  - Encapsulate domain-related activities
  - Decompose processes hierarchies or networks as appropriate
  - Explicitly define the relationships among processes

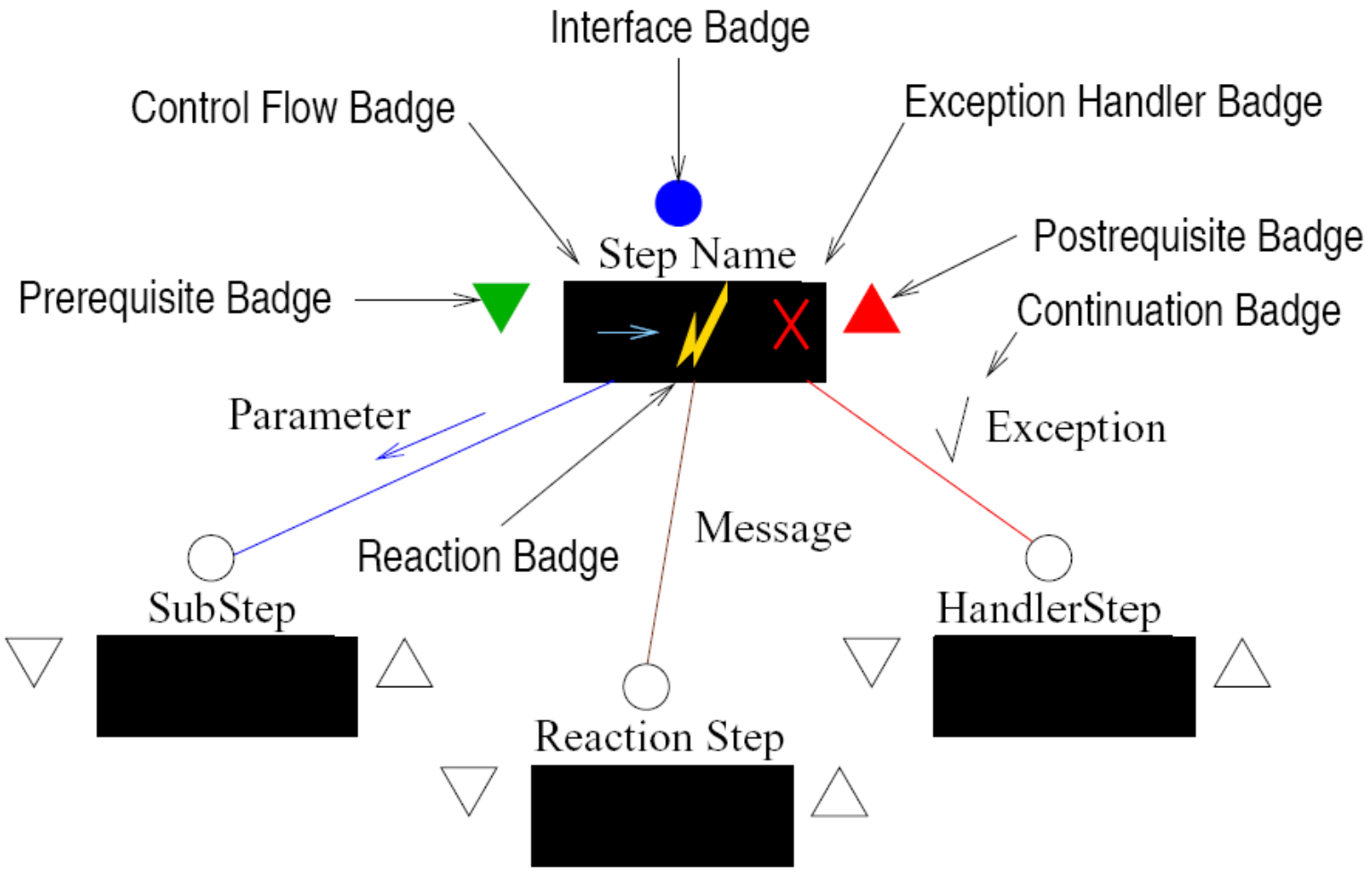
Construction

- Implementation of components
  - Process programming
    - Code the process details
    - Step typically the analog of function/procedure
    - Explicit instructions and control
  - Process modeling
    - Incomplete descriptions (ie, models are inherently incomplete)
    - Variety of modeling mechanisms
      - ✓Petri-nets
      - ✓Finite state machines (variants: state charts)
      - ✓Data flow diagrams
      - ✓Goal directed models
- Construction
  - Typically dynamic execution and support
  - Don't know of any static build process
- Two examples: Little-JIL and Interact/Intermediate

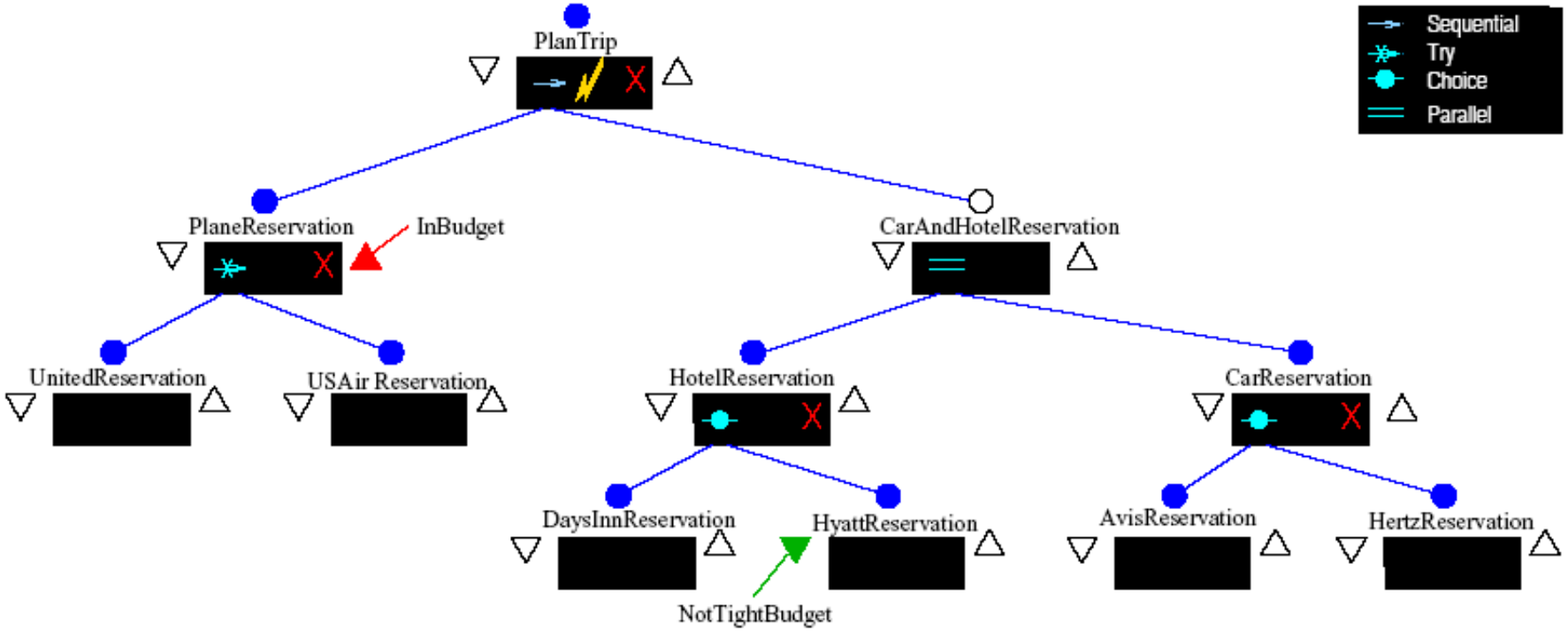
Osterweil's Little-JIL Features

- 4 kinds of steps
  - Sequential – left to right order
  - Parallel - simultaneously
  - Try – left to right, stopping when one competes successfully
  - Choice – arbitrarily decides which
- Requisites
  - Prerequisite – must complete prior
  - Postrequisite – must be completed after
    - Somewhat similar to Inscape's obligation
- Exceptions and Handlers
  - Provide reactive control
  - Unhandled exceptions propagate up the execution tree
- Messages and reactions
  - Also reactive control, but no propagation
- Parameters – communication of information
- Resources – required during a step execution

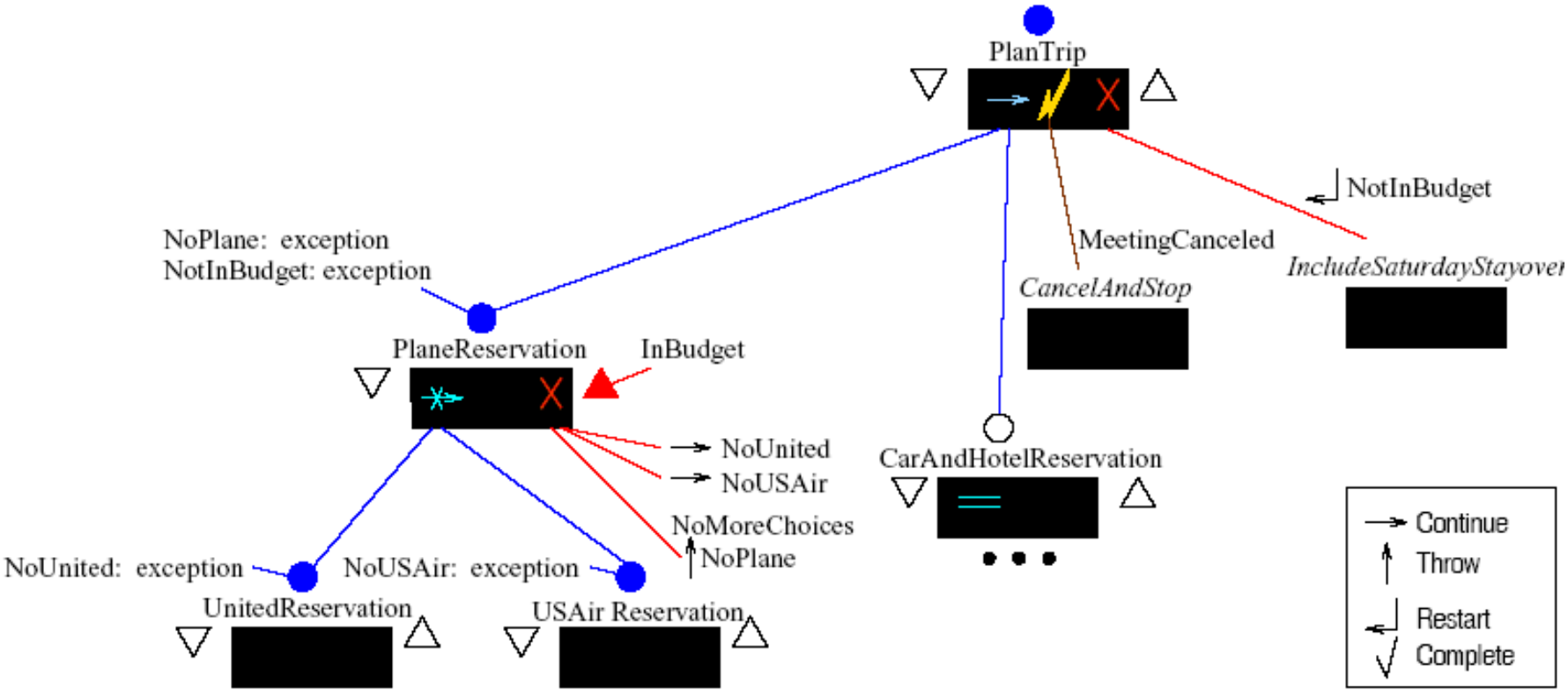
Osterweil's Little-JIL Features



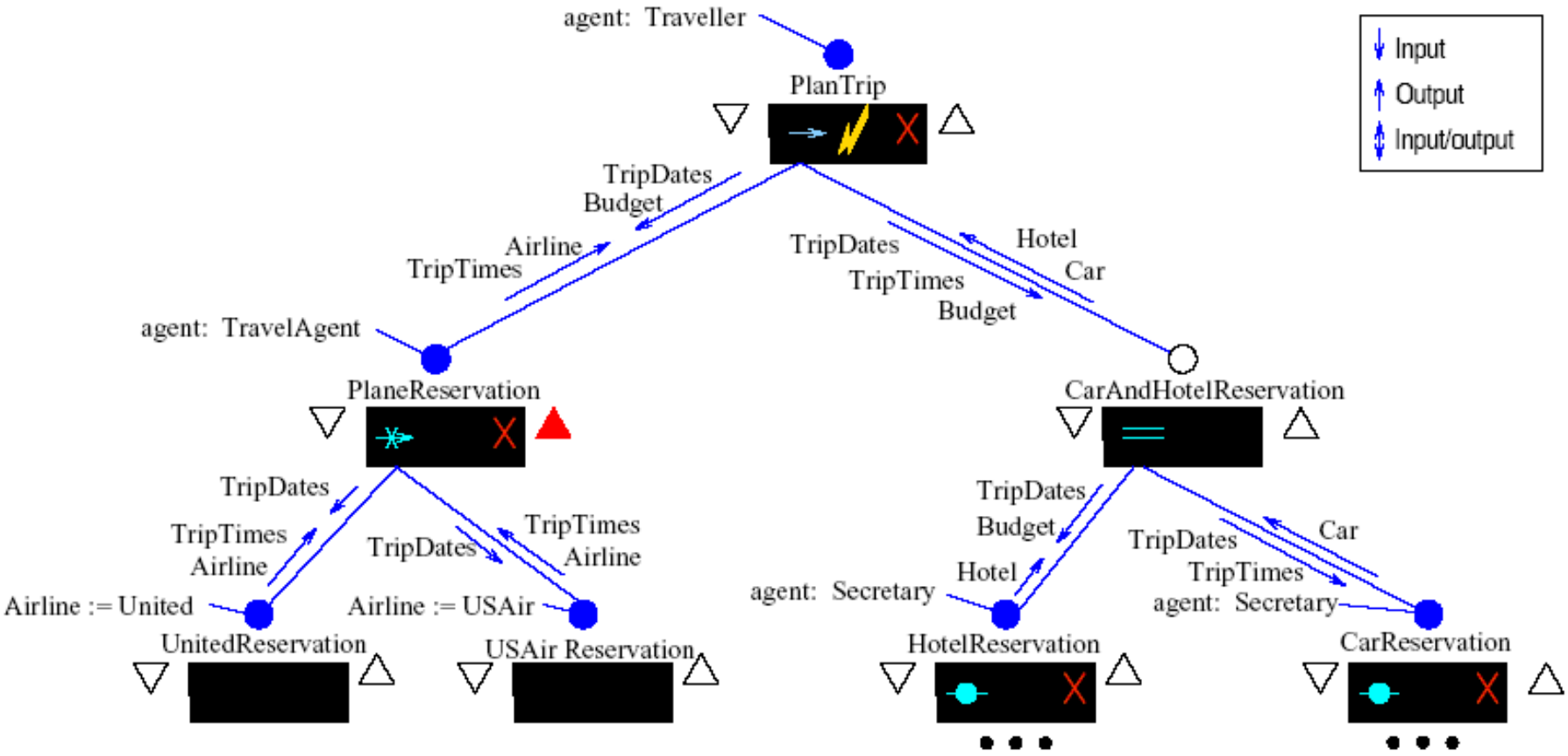
Proactive Control (Steps, Requisites)



Reactive Control (Exceptions, Messages)



Process Data Flow



Interact/Intermediate

- A goal-directed process modeling language
- Philosophy:
  - To maximize concurrency of process activities
  - To minimize direct control of the human element in the process
- Emphasis:
  - Specifying assumptions and goals of various process activities
  - Leaving details of the activities implementation to the enactor
  - Define some details of the enactment structure when desired
- Critical design considerations
  - Dynamism – software processes extremely dynamic
  - Reflectivity – activities are dependent on state of
    - Artifacts, process, project and organization

Interact/Intermediate: Features

- Activity – basic process fragments
  - Signature: name and typed parameters
  - Assumptions – (ie, pre-conditions)
  - Internal structure
    - Primitive – determined at enactment time by the enactor
    - Sequence < . . . > in specified sequence
    - Set { . . . } in arbitrary order
    - Selection (guard, structure) when guard, structure
  - Set of results, one or more of which may be achieved
    - Goals achieved (ie, post-conditions)
    - Obligations - goals that must eventually become true
- Data – basic, enumerated, structured

Interact/Intermediate: Features

- Enactment control
  - Implicit: partial order defined by assumptions and goals
    - Goals provide the state that can be used to satisfy assumptions
    - Determine an implicit ordering of the activities
    - Can only instantiate an activity when its assumptions have been met
  - Explicit: normal enactment control
    - Range from primitive (where left up to the enactor)
    - To specified, but preferably under specified, structure
    - Human enactor elaborates activity using supporting environment
  - Explicit: abnormal enactment control
    - When exceptions happen
  - Explicit: external constraint on beginning or end of enactment

Interact/Intermediate: Example

```
activity Integrate ( )
  preconditions { Release-Approved(Tool-Release-Board) }
  {

  }
  results
    <
      ( postconditions {
            approvedset = { tool t | tool-approved(t) },
            exportset = exportset + approvedset,
            tools-released(exportset) }  ,
        obligations { }
      ) ,
      ( postconditions { rejectset = { tool t | tool-rejected(t) } }  ,
        obligations { for each tool t in reject-set: modify-tool(t) }
      )
    >
```

Interact/Intermediate: Example

```
for each  tool t  in  {tool t | submitted(t) }
until  Current-Time == Deadline:
   <
      Determine-Dependencies(t, dependencies) ,
      let  testset' = testset + t ,
      Build(testset', result) ,
      (  result == false, tool-rejected(t)  )  ,
      (  result == true,
         <
            <for each  person P
                  in  {person p | owner[t1] == p & t1  in  dependencies }:
                     bind  Evaluate(t, t1) to  P
            > ,
            Await-Acceptance/Rejection(t)
         >
      )
   >
```

Documentation

- Same lessons as product documentation
  - Shared understanding of what to do
- 5ESS lessons: inappropriate process documentation
  - process description often describe how to write document
  - product production vs. management tasks are all the same level
  - process input-output reflects benchmarks not just I/O
  - process descriptions are too detailed
    - they define how?
    - as well as what?
  - too much text
  - processes too detailed
  - process description vs process prescription
- Important problem
  - Finding the right level for process descriptions
  - Including necessary information in the descriptions

Other Commonalities with Products

- Deployment and Maintenance
  - Basically the same problems
  - But little likelihood of automation
  - Too much unformalized (unformalizable?) knowledge
- Version Management
  - Identical
- Teamwork
  - Identical