Exploring the Impact of Employee Turnover on the Effectiveness of Software Development Team Archetypes

Levent Yilmaz
M&SNet: Auburn M&S Laboratory
Computer Science & Software Engineering
Auburn University, Auburn, AL 36849

Aim

- to present a strategy for developing process simulations based on the science of organizations,
- to promote the significance of modeling cooperation mechanisms in software development
- To explore the extent of the impact of turbulence (i.e., employee turnover) on the effectiveness of software development under various team archetypes in small organizations.
Motivation: Exploring the impact of organizational dynamics
Organizational concepts for software process simulation
Team-RUP: An agent-based framework for cooperative team behavior simulation
A case study: Effects of environmental turbulence on development organizations
Conclusions

Plan

Motivation: Organizational Dynamics and Software Development

Software production methods are
- enacted via interactions of software teams that cooperate to build software (Sawyer 1994).
- carried out by teams of people that have to be coordinated within an organizational structure.

Human organizations, including software development organizations,
- (1) continually acquire, manipulate, and produce artifacts through joint cooperative activities and
- (2) are comprised of multiple distributed agents that exhibit collective properties by communication, collaboration, and coordination.
Motivation (cont’d)

- a better balance between foundational knowledge of engineering of software on the one hand, and human, social, and organizational dynamics on the other hand is needed.
- Our work on the organization-theoretic perspective of process modeling entails characterizing:
  - the components of organizational design and
  - types of organizational paradigms (team archetypes)

Plan

- Motivation: Exploring the impact of human, team, and organizational dynamics on software development effectiveness
- Organizational concepts for software process simulation
- Team-RUP: An agent-based framework for cooperative team behavior simulation
- A case study: Effects of environmental turbulence on development organizations
- Conclusions
Organizational Concepts for Software Process Simulation

Organizational Design

- Structure of the organization includes a set of relations involving:
  - its members (i.e., agents),
  - skills and resources,
  - task-resource, task-skill, resource access, task-precedence, and task assignment are among the fundamental elements of organizational structure.
Organizational Behavior Moderators

Behavior moderators or stressors such as time pressure, deadlines, turnover, trust, reward mechanisms, influence the observed behavior of a software process.

Organizational Strategy and Paradigms

A strategy is realized in terms of cooperation mechanisms that require coordination and collaboration among the members of the organization.
- planning, rules, mutual agreement, and hierarchy.
Organizational Paradigms


<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Coordination</th>
<th>System regulation</th>
<th>Priorities</th>
<th>Decision making</th>
</tr>
</thead>
<tbody>
<tr>
<td>closed</td>
<td>traditional authority</td>
<td>negative feedback, deviation attenuating</td>
<td>stability, group; secure continuity</td>
<td>formal, top-down by position</td>
</tr>
<tr>
<td>random</td>
<td>innovative independent initiative</td>
<td>positive feedback, deviation amplifying</td>
<td>variety, individual; creative innovation</td>
<td>informal, bottom-up, by individual</td>
</tr>
<tr>
<td>open</td>
<td>adaptive collaborative process</td>
<td>combined feedback, flexible responsiveness</td>
<td>stability and change, group and individual; adaptive effectiveness</td>
<td>negotiated, consensual, by group process</td>
</tr>
<tr>
<td>synchronous</td>
<td>efficient harmonious alignment</td>
<td>shared programming, efficient uniformity</td>
<td>harmony, manual identification; efficiency of coordination</td>
<td>unnegotiated, predefined, implied by vision</td>
</tr>
</tbody>
</table>

Task-Environment Model

- Refers to the environmental and task characteristics that affect the performance of the organization.
- Characteristics include type, size, rate of change, uncertainty, interdependence, complexity, and granularity of tasks.
Motivation: Exploring the impact of human, team, and organizational dynamics on software development effectiveness

Context: Agile software development with RUP

Organizational concepts for software process simulation

**Team-RUP:** An agent-based framework for cooperative team behavior simulation

A case study: Effects of environmental turbulence on development organizations

Conclusions
Project Configuration → C = array of integers.
Possible Problem Facets →
{(ci, cj) ∈ C x C | i < j}

Comparing (for order) c_h and c_i or comparing c_i and c_k is analogous to performing some task associated with (c_i, c_j). If (c_i, c_j) is an inversion that is removed as a result of the comparison, the task is primary. Otherwise, it is secondary. Requirements, therefore, are sets of inversions.
Why Shell Sort?

- Inversions removed in phases corresponds to iterative development
- Any later phase of Shell sort does not undo the “work” of earlier phases (no throw-away work-products)
- Inversions introduced after the start of the algorithm but before the last phase will still be removed (copes with requirements change)
- Provides a framework for sorting but relies on secondary sorting algorithm to do the actual sorting
  - Secondary algorithms allow us to encode team behavior

Team-RUP Behavior Archetypes

<table>
<thead>
<tr>
<th>Degree of autonomy in collaboration</th>
<th>Degree of concurrency in coordination</th>
<th>Low (Top-down task allocation &amp; problem solving)</th>
<th>High (Bottom-up task allocation &amp; problem solving)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low (Linear)</td>
<td>Synchronized</td>
<td>Agile</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>High (Concurrent)</td>
<td>Asynchronous</td>
<td>Autonomous</td>
<td></td>
</tr>
</tbody>
</table>
Team Behaviors (list sorting)

- The set of ways to sort a list is infinite but there are relatively few useful strategies. Can our four team behaviors be patterned after some of these useful strategies?
  - Autonomous (bottom-up, concurrent): Merge sort
  - Asynchronous (top-down, concurrent): Quick sort
  - Agile (bottom-up, linear): Insertion Sort
  - Synchronized (top-down, linear): Heap Sort
- Won’t we just be (re-)finding the best sorting algorithm?

Variation on a Theme

- Team-RUP uses idea behind sorting algorithm, NOT the algorithm itself.
- Sorting strategy provides a schedule of activities but does not determine how the teams manage the actual sorting.
- How actual sorting takes place depends on parameter settings (e.g., workload) and qualitative observations concerning team types.
Asynchronous Team Behavior

- Concurrent and top-down
- Step-wise refinement of high-level requirements into a solution
- Systems Engineering approach: concept → production → operation
  
  (INCOSE)
- Not very adaptive
  - Concurrent coordination strategy slows the organization as a whole in modifying its approach
  - Turbulence has a more global impact on a project when a top-down approach is utilized. The plan ceases to fit the task (as well).

Asynchronous Team (Quick Sort)

- Design Manager tells a single team to pick a pivot and partition.
- Design Manager tells teams to perform the same actions on each partition.
- So forth and so on until partition drops below the workload threshold. Then sorted by a single team.
Cooperation Strategy for Asynchronous Teams

Agile Behavior

- **Linear** (as opposed to concurrent) and bottom-up
- **Linear** - “Business people and developers must **work together daily** throughout the project”
- **Bottom-up** – “…**early and continuous** delivery of software”
- Within a process framework, this is the opposite of the traditional hierarchy. As such, it shares close relationship with Constantine’s Random archetype.
Agile (Insertion Sort)

The Design Manager tells teams to insert their values.
- Teams communicate to preserve sub-list order
- Design Manager keeps up with which team is managing which values.

Synchronized Behavior

- Linear and top-down
- Most structured behavior
- Bureaucracy of experts
- Well-defined workflow
- Division of labor leads to confounding effects in terms of efficiency and effectiveness
  - Specialist becomes very good at specific task
  - Blockages in workflow can result if specialist stalls
- Ability to adapt is confounded as with Autonomous team behavior
Synchronized (Heap Sort)

- Teams manage contiguous cells of the heap array (areas of expertise).
- Teams are aware of each other’s responsibilities independently of the design manager.
- Expertise-related tasks bind to values when they enter a team’s region.
  - Can block the workflow if not completed and heapify operation is eminent.
  - Greater experience = faster completion of these tasks.

Implementation

- Agents composed of Java objects
- Plugs into the Repast simulation engine
- Time advances of fixed width
  - 1 workday = 15 time ticks
- RUP iteration = 300 ticks = 4 five-day weeks
  - 60 time-boxed iterations = 5 years
  - 1 iteration as warm up
  - For a single team with medium experience (0.5) to implement (sort by self) an array of 3 elements, it takes approximately 2 days.
    - $E[\text{Implementation time}] = \text{componentSize} / \text{experience} \times \text{workTimeParameter} \ (\text{current setting is 5 ticks})$
The RePast Model

- Developed at the University of Chicago by Sallach, Collier, Howe, and North
- Open source toolkit for modeling social agents
- Supports a variety of languages including Java, C#, and Python
- Includes a variety of templates and tools for agent modeling and graphical representation
- Includes a discrete-event scheduler

Model Parameters

- Complexity: Percentage of inversions present out of total possible
- Product Scope: Size of array
- Number of Teams
- Requirements Stability: The percentage of array locations that are unchanged between RUP iterations. Percentage increases with each iteration.
- Workload
- Turbulence
  - External turbulence = 100% - (Requirements Stability)
  - Internal turbulence: Employee turnover
Members of *Engineering* teams separate from organization according to a “firing process”

- Firing algorithm based on a moving average of milestone performance evaluations that occur at the end of each iteration as well as history of consecutive failures.
- Firing decreases team experience and can cause a team to stall.
Three variables, each of which takes on a value between 0 and 1, influence the level of internal turbulence.

- The base rate parameter $B$ represents the degree to which a company is downsizing.
- The sensitivity to performance parameter $S$ determines the degree to which an employee’s performance affects his chance of separating from the company.
- This number, of course, has little meaning without the performance variable $P$. Performance is measured at the team level. Initially, the performance for each team is 0.5.

Suppose the requirements array is to be $n$-sorted by $k$ teams during iteration $i$, and $m$ inversions should be removed by this operation. The expected number $\beta_t$ of removed inversions for team $t$ is defined as follows:

$$\beta_t = \frac{m}{k}$$
Employee Turnover

- Assuming team \( t \) actually removed \( \alpha_t \) inversions during iteration \( i \), the milestone performance \( \delta_{iti} \) of \( t \) during \( i \) is defined by

\[
\delta_{iti} = \frac{\alpha_t}{\beta_t}
\]

- While milestone performance is measured each iteration, team performance is computed over a window consisting of multiple iterations, and involves a moving average of the milestone performance measures observed during these iterations.

Employee Turnover

- We define the failure history \( f_{iti} \) of a team \( t \) for iteration \( i \) as follows

\[
f_{iti} = \begin{cases} 
0 & \delta_{iti} \geq 1; \\
\left( f_{iti-1} + 1 \right) & \delta_{iti} < 1.
\end{cases}
\]

- For a performance window of size \( w \), we define the team performance \( P_{tk} \) of team \( t \) at the end of iteration \( k \) via

\[
P_{tk} = \frac{1}{w} \sum_{i=1}^{w} \frac{\delta_{iti}}{f_{iti} + 1}
\]
Employee Turnover

During a performance evaluation, teams with a performance measure greater than a predefined threshold (0.3 in the current implementation) are immune to employee turnover.

Of the teams below this threshold, the team with the smallest performance rating undergoes subjective appraisal.

Employee Turnover

When a real-world team is evaluated, the evaluator has incomplete information concerning a team and is influenced by various biases.

To reflect this fact, the separation of an employee from the company in the Team-RUP framework is a probabilistic event.
Employee Turnover

- The probability $F_t$ that some member of team $t$ will separate from the company at the end of iteration $i$ is given by the following equation:

$$F_t = B + S \cdot (1 - R_t)^3$$

- Internal turbulence $T$ can then be quantified as any linear combination of the base rate and sensitivity to performance.

$$T = \alpha B + \beta S$$

Validation
Definitions

- **Product Size**: The number of inversions in the array after warm up
- **Productive tick**: A tick in which team is not idle.
  - Measured on a per team basis
- **Ticks**: Number of time ticks occurring after warm up
- **Unit of Functional Content**: Cell of Partition
  - Planned
  - Completed

```
One unit
```

Metrics

- Based on Practical Software Measurement v. 3.1a (Office of the Under Secretary of Defense for Acquisition and Technology)
- **Productivity** = \( \frac{\text{removedInversions}}{\text{Ticks} \times \text{numAgents}} \)
- **Staff Utilization** = \( \frac{\text{ProductiveTicks}}{\text{Ticks} \times \text{numAgents}} \)
- **Timeliness** = \( \frac{\text{completedFunctionalContent}}{\text{plannedFunctionalContent}} \)
- **Quality** = \( 1 - \frac{\text{remainingInversions}}{\text{productSize}} \)
Validation of Coordination

- Validation Point: Asynchronous systems are less predictable than synchronous systems (Shamsi, Chu, and Brockmeyer, 2005).
- Experiment: Looked at standard deviations of response values over multiple replications and factor settings.

Agile Validation

- Random (Agile) teams are not inherently efficient (Constantine, 1993).
- Quantify efficiency as ratio of productivity to staff utilization.
- Experiment: Consider efficiency of Agile teams over multiple replications and factor levels.
**Plan**

- **Motivation:** Exploring the impact of human, team, and organizational dynamics on software development effectiveness
- **Context:** Agile software development with RUP
- **Organizational concepts for software process simulation**
- **Team-RUP:** An agent-based framework for cooperative team behavior simulation
- **A case study:** Effects of environmental turbulence on development organizations
- **Conclusions**

**Agility Test**

- **Question to address:** Are the claims of the agile community valid?
- **Experiment:** Compare team behaviors in terms of their ability to cope with the combination of internal and external turbulence.
Team Size Test

- Question: How well do small organizations cope with turbulence?
- Experiment: Compare teams of different sizes across varying levels of turbulence
**Plan**

- Motivation: Exploring the impact of human, team, and organizational dynamics on software development effectiveness
- Context: Agile software development with RUP
- Organizational concepts for software process simulation
- Team-RUP: An agent-based framework for cooperative team behavior simulation
- A case study: Effects of environmental turbulence on development organizations
- **Conclusions**

**Findings**

- Concurrent teams are most affected by turbulence.
- With its rigid structure, the Synchronized behavior is the least suited for adaptation to changing requirements with firm deadlines.
- Process agility is a valid and useful counterbalance to the inevitable change involved in most real-world software projects.
- Small organizations should consider adopting a software process that encourages agile behavior.
- If greater independence among teams is necessitated by a particular project, a large organization will perform significantly better than a smaller one.
Future Work

- Replace *engineering team* agents with individual developer agents
- Cognitive factors such as stress, trust, and competition
- Extend agent hierarchy to entire development organization (at present, limited to construction division)
- Multiple requirements arrays

Questions?