Requirements for Distributed Teams

• Processes and planning
• Requirements difficulties and approaches

DSD Issues and Risks

• Key Problem: coordination at a distance
  i.e., getting people to do the right task the right way at the right time
• Key risk factors:
  – Restricted communication, flow of information
  – Different organization, language, culture
  – Lack of visibility into what remote teams are doing
• Potential difficulties:
  – Different views of the problem (requirements)
  – Different views of what the process is supposed to be
  – Misunderstanding of what remote teams are doing
  – Difficult to detect and correct problems
  – Difficult to manage synchronize the work
  – Difficult to detect and correct slips in schedule
## Summary Co-located vs. DSD

<table>
<thead>
<tr>
<th>Co-located Development</th>
<th>DSD Risks*</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Free flow of information through informal means</td>
<td>• Restricted flow of information, mostly formal</td>
</tr>
<tr>
<td>• Shared process view</td>
<td>• Possibly different process views</td>
</tr>
<tr>
<td>• Clear idea of expertise, responsibility</td>
<td>• Unclear idea of expertise, responsibility on remote teams</td>
</tr>
<tr>
<td>• Common culture eases understanding</td>
<td>• Possible misunderstandings due to cultural/language differences</td>
</tr>
<tr>
<td>• Understand relationships</td>
<td>• Vague or incorrect understanding of relationships</td>
</tr>
<tr>
<td>– People to tasks</td>
<td></td>
</tr>
<tr>
<td>– Task interdependencies</td>
<td></td>
</tr>
</tbody>
</table>

*Standardizing the process helps mitigate these risks a people fill roles with well-defined responsibilities.

When will Foober be ready?  
Who is working on it?  
Are there any problems?  
Who should I tell about the problem I encountered in the Bazbot?
Incremental Development Over Time

- Acts as a feedback loop with a calibration point at each delivery
  - Allows cross checking of assumptions, understanding
  - Early check if remote sites are doing what is expected
  - Early check for communication effectiveness
  - Allows plan adjustments at each increment

Well-defined Process Benefits

- Process should also be relatively formal
  - Written down in detail
  - Required for all of the distributed sites
- Well-defined process clearly specifies
  - The artifacts to be produced
  - The set of activates that must be performed
  - The set of roles
  - The relationships
    - Which roles perform which activities to produce which artifacts
    - The order of activities
    - Which artifacts are needed as input to produce which other artifacts
Well-defined Process Benefits

- Helps address risks
  - Everyone has common definition of the process
  - Assigning roles clearly defines responsibilities
  - Helps make clear what people should be working on
  - Helps make clear when a task is finished
- Should help answer for individuals the questions
  - Is this my job?
  - What do I do next?
  - Am I done yet?
  - Did I do a good job?

From Process to Plan

- Process definition manifests itself in the project plan
  - Process definition is an abstraction
  - Many possible ways of implementing the same process
- Project plan makes process concrete, it assigns
  - People to roles
  - Artifacts to deliverables
  - Activities to tasks over time
- For DSD, it is essential that distributed teams agree on the project plan
Project Plan

- Minimal plan contents
  - Risks and mitigation strategies
    - Evolves with progress and understanding
  - Tasks to be performed
  - Person(s) assigned to roles and tasks
  - Deadline for each task
  - Sequencing among tasks
    - Task dependencies
    - Development plan
- Usually owned by team leads (one at each site)
- Updated as project proceeds

How to Build a Plan

- Look for major milestones then work backwards
  - Identify intermediate deliverables
  - Identify any precedence constraints
  - Decompose into units of work (work breakdown)
  - Allocate roles
- Planning proceeds by successive approximation
- Strategy: focus first on risk mitigation
  - Early successes: start with simple tasks
  - Identify risky parts, prioritize mitigation activities
  - Build feedback in
    - Reviews
    - Synchronization points (e.g., builds)
- Opportunity to learn Gantt charts
  - Use a free web-based app (e.g., smartsheet)
Roles and Responsibilities

• Well defined roles provide a badly needed structure
  – Define who is responsible for what
  – Gives guidance for expected expertise
• Relations between roles tell you
  – Who needs to talk to each other (e.g., shared responsibility, handoff, etc.)
  – What you need to be talking about
  – Provides bases for forming professional relationships
• Upshot: in DSD it is critical that
  1. Roles and their responsibilities are clearly defined
  2. Well defined lines of communication are established between roles at different sites
  3. People consistently perform their role’s responsibilities

Summary

• Common project plan is key to coordination
  – Clear definition of roles and responsibilities
  – Clear dependencies between tasks hence, what needs to be done next
  – Provides basis for tracking progress
• Just one part of necessary communication!
  – Teams must agree on project plan but…
  – Still can have misunderstanding about meaning of plan
  – Still may go off track
    • Must detect and correct as soon as possible
    • This is not easy
  – Plan must be continuously checked, updated
Managing Software Requirements in DSD

What is a “software requirement?”

• A description of something the software must do or property it must have
• The set of system requirements denote the problem to be solved and any constraints on the solution
  – Ideally, requirements specify precisely what the software must do without describing how to do it
  – Any system that meets requirements should be an acceptable implementation
Requirements Phase Goals

- Because requirements decisions come first, important to get them right
- Only three goals
  1. Understand precisely what is required of the software
  2. Communicate that understanding to all of the parties involved in the development (stakeholders)
  3. Control production to ensure the final system satisfies the requirements
- Sounds straight-forward but hard to do in practice

What makes requirements difficult?

- Comprehension (understanding)
  - People don’t (really) know what they want (…until they see it)
  - Superficial grasp is insufficient to build correct software
- Communication
  - People work best with regular structures, conceptual coherence, and visualization
  - Software’s conceptual structures are complex, arbitrary, and difficult to visualize
- Control
  - Difficult to predict which requirements will be hard to meet
  - Requirements change all the time
  - Makes planning unreliable, cost and schedule unpredictable
- Inseparable Concerns
  - Many requirements issues intrinsically interdependent
  - Difficult to apply “divide and conquer”
  - Must make tradeoffs where requirements conflict
Additional Risks of DSD

- DSD tends to aggravate existing difficulties
- Comprehension
  - Different languages, cultures, expectations
  - Greater risk of ambiguity, misunderstanding
- Communication
  - Constrained communication: limited bandwidth, language, culture, invisible stakeholders
  - More difficult to negotiate common understanding, may miss issues
- Control
  - Less feedback, often delayed
  - Easy to get out of synch and not know it
  - Easy to lose track of the effects of changes
- Inseparable concerns
  - Difficulty making independent work assignments among sites
  - Conversely, easy to have inadvertent side effects on other code

Sources of Errors in DSD

Lessons Learned in Distributed Software Development – Komi-Sirvio, Tihinen

CIS 423/523 © S. Faulk
In Planning: View as Risks

• Risk: requirements are misunderstood
  – Problem: There is a misunderstanding of exactly what the stakeholders require or the distributed teams have a different understanding of one or more requirements.
  – Mitigations: ?

• Risk: missing requirements
  – Problem: One or more requirements are not identified or the distributed teams have somewhat different sets of requirements.
  – Mitigation: ?

• Risk: requirements change
  – Problem: During development, one or more requirements change or, equivalently, a team’s understanding changes.
  – Mitigation: ?

Risk Mitigation Strategies

• Build risk mitigation into the project’s software process
  – Requirements changes occur throughout development
  – Must address at all stages of process

• Requirements are missing, misunderstood
  – Requirements exploration with stakeholders (customer)
  – Early modeling: prototypes, mockups
  – Precise documentation
  – Careful review
  – Incremental delivery
  – Clear responsibilities for requirements tasks, products

• Requirements change
  – Consider the effects of changes in advance
  – Software design for robustness, ease of change
  – Explicit processes for managing change (baseline, change request, triage, approval, dissemination, tracking, etc.)
Requirements Process Components

- Process components – integrate into project plan
- Activities: deploy explicit requirements activities
  - Requirements exploration and understanding
  - Requirements negotiation (and explicit signoff)
  - Requirements specification
  - Requirements verification and validation (feedback)
  - Change management for distributed team
- Artifacts: provide vehicles for capture, communication & assessment, e.g.,
  - Prototypes, mock-up, story board, use cases
  - Common requirements specification
  - Reviews
- Roles: create clear responsibilities for activities, artifacts, and communication, e.g.,
  - Analyst: exploration, customer interaction, negotiation
  - Requirements Engineer: specification
  - Reviewer: verification and validation

Process View

<table>
<thead>
<tr>
<th>Activities</th>
<th>Artifacts</th>
<th>Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Prototypes, use cases, etc</td>
<td>Customer</td>
</tr>
<tr>
<td>• Elicitation &amp; negotiation</td>
<td>ConOps</td>
<td>Analyst</td>
</tr>
<tr>
<td>• Specification</td>
<td>Formal SRS</td>
<td>Reviewer</td>
</tr>
<tr>
<td>• Verification and validation</td>
<td>QA plan (reviews, tests)</td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td>Module structure</td>
<td>Architect</td>
</tr>
<tr>
<td>• Design for distribution</td>
<td>Module interface specs.</td>
<td></td>
</tr>
<tr>
<td>• Design for change</td>
<td>Active review</td>
<td></td>
</tr>
<tr>
<td>• Verification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detailed Design/code</td>
<td>Test cases</td>
<td>Coder</td>
</tr>
<tr>
<td>• Verification and validation</td>
<td></td>
<td>Tester</td>
</tr>
<tr>
<td>System Integration/test</td>
<td>Test cases</td>
<td>Customer</td>
</tr>
<tr>
<td>• Validation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change Management</td>
<td>Change Request</td>
<td></td>
</tr>
</tbody>
</table>
Assignment

- Use the ConOps section of requirements to capture the proposed system capabilities
- Look at example ConOps document for relevant sections

Questions?