Introduction to Distributed Software Development

Stuart Faulk
CIS
University of Oregon

Lian Yu
School of Software and Microelectronics
Peking University

Outline

- Administration
  - Course purpose and format
  - Projects and teams
- Lecture: Introduction to Distributed Software Development (DSD)
- Lab: Team formation
- Lecture: Role of software engineering
- Lab: Set up accounts

Course Purpose

- Understand the challenges of globally distributed software development (DSD)
- Understand how software engineering methods can help
  - Some familiar techniques applied in new way
- Apply the methods on a small project simulating DSD
- Work with students from other countries, make friends
- Practice international English

© S. Faulk 2010
Course Format

- Mix of lectures and exercises - approximately
  - 50 min lecture, break
  - 50 min lab exercise, discussion
  - 50 min lecture
  - 60 min lab exercise
- Lectures: foundations and background
- Team exercises: simulate some of the problems of DSD
- Learn SE skills by encountering and overcoming DSD problems
- Project reviews and presentations: critique and guidance
- Learn teamwork across cultural boundaries
- Work collaboratively at a distance in lab setting

Weekly Schedule

- Day 1: Introduction to DSD and role of SE
- Day 2: Choosing and modeling software processes, project planning
- Day 3: Requirements and managing change
- Day 4: Architecture I: module structure
- Day 5: Architecture II: abstract interface design
- Day 6: Architecture III: design for extension and contraction, process structure (threads)
- Day 7: Testing (Prof. Lian Yu)
- Day 8: Project results and course review

Introduction to Distributed Software Development

Purposes of DSD
Issues and Risks Introduced
The Role of Software Engineering
Growth of DSD

- Increasingly used by companies of all but the smallest size
- Outsourcing growth by an order of magnitude over past few years
- Over half of the Fortune 500 companies
  - Software industry: Google, Microsoft, IBM, etc.
  - Other industry: telecom, cell phone, etc.
- If you work in industry, you will likely work in a distributed team

Sample of Development of 52 Major Projects at Avaya Corp. 2006

Approximately 75% multi-site development and growing

Rationale for DSD

- Expanded pool of trained workforce
- Necessity of getting closer to customers and using locality specific expertise to customize/localize products
- National policy in some countries requiring suppliers to locate facilities in that country as a contract condition
- Difference in development costs
- Promise of round-the-clock development that could lead to shorter intervals
Benefits Often Come at a Cost

- With DSD benefits come increased risks compared to similar co-located developments
- Schedule delays — same work takes longer
- Higher risk of failure
- Reduced product capabilities
  - Decreased functionality, qualities
  - Doesn’t meet some customer requirements
- Increased cost
  - May cost more in spite of lower labor costs
  - Schedule delays and rework increase costs

Working Definition

**Distributed Software Development (DSD):** teams in geographically distant locations collaborate to produce the work products of a software development
- Synchronize in all phases of the life cycle
- Collaborate on artifacts from requirements to code
- Coordinate activities among members of distributed teams
Observed Difficulties (1)

- Nature of a software project
  - Software development produces a set of interlocking, interdependent work products
    - E.g. Requirements → Design → Code
  - Implies dependencies between tasks
  - Implies dependencies between people
- Successful development requires effective coordination between people and tasks!
  - Must coordinate work (need product A to produce product B)
  - Must coordinate schedule (must finish A before starting B)
  - Must coordinate people (person P has expertise need to produce A but is busy)

Observed Difficulties (2)

- Key property distinguishing DSD from co-located development
  - "The key phenomenon of DSD is coordination over distance." — J. Herbsleb (2007)
- All software projects require coordination
- Suggests that coordination at a distance is different
- Managing these differences is a central issue in DSD

How is DSD Different?

- In co-located projects, people develop informal ways of coordinating work
  - Shared process view (implicit or explicit)
  - Common vocabulary, viewpoint
  - Clear idea of expertise, responsibility
  - Free flow of information through informal channels
  - Common language, culture, backgrounds helps avoid miscommunication
  - Relatively good understanding of relationships
    - People to tasks
    - Task dependencies
    - Professional and social
DSD is Different…

- In DSD many of the mechanisms for coordinating work are absent or disrupted
  - Much less communication
    - Temporal distance
    - Socio-cultural distance, e.g., language
  - Spontaneous communication declines rapidly with distance
  - Less effective communication
    - Fewer overlapping work hours
  - Low bandwidth links (e.g., email and other asynchronous)
  - Lack of awareness
    - Lack context hence knowledge of history, relationships
  - What people are doing day to day, concerns, availability
  - Incompatibilities
    - Differ in tools, processes, work products
  - Some issues are observed with even small distances (e.g., 30 meters)

Software Development Problems

- Manifests as problems in coordination and control of software development
  - Difficult managing development resources (schedule, personnel, budget)
  - Difficulty establishing requirements (eliciting, understanding, negotiating)
  - Difficulty effectively distributing work
  - Difficulty detecting and correcting conflicting assumptions
  - Difficulty detecting and correcting slips in schedule
  - Difficulty managing change (especially requirements)
- Similar to traditional SE problems, but more so
  - Work takes longer
  - Requires more effort

Useful to View as Risks

- Examples of increased project risks due to communication and control difficulties
  - Building the correct software (behavior or qualities)
    - Fully understand (or misunderstand) the requirements
    - Developers build the wrong thing
    - Inadequate verification, validation
    - Decomposing the work into work assignments that together address behavioral and developmental goals
    - Ensuring each team builds what is needed by others
  - Managing development (budget, schedule, personnel)
    - Balancing workload, skills across sites to meet developmental and quality goals
    - Developing common understanding of who should be developing what in which timeframe
- We will examine risks in detail
- Fundamental issue is how to mitigate DSD risks
Questions?

Assignment
- Prof. Jeff Stolle: team-building exercise

The Role of Software Engineering
Brief overview of relationship of SE to managing developmental risks
Outline

- Purpose of software engineering
- Software engineering areas addressing DSD risks
  - Process management
  - Requirements negotiation and specification
  - Design for collaborative development
- assembla demo: our development environment
- Exercise: introduction to assembla

Objectives

- Key property distinguishing DSD from co-located development
  - "The key phenomenon of DSD is coordination over distance." – J. Herbsleb (2007)
- Coordination problems lead to development risk
  - Delivered late
  - Higher cost
  - Reduced capability
  - Higher likelihood of failure
- Our goal is to control the risks of DSD to produce the right software on time with available resources

Purpose of SW Engineering

- The purpose of software engineering is to gain and maintain intellectual and managerial control over the products and processes of software development
  - "Intellectual control" means we are able make rational technical choices based on an understanding of the downstream effects of those choices
  - Managerial control means we are able to make rational choices about development resources (budget, schedule, personnel) to deliver software on time and within budget
Meaning of *Intellectual Control*

- Software development progresses through a sequence of decisions
  - Decisions about requirements (e.g., tradeoffs, priority)
  - Design decisions (e.g., first decomposition)
- Earlier decisions affect the difficulty of later decisions
  - Ensuring that we end up with the desired properties requires making the right decisions in the right order
  - E.g., cannot add properties like security late in the game, Windows demonstrates this vs. OSX (Unix private address space)
- Being in control means we can:
  - Decide in advance the functional and non-functional requirements the software should satisfy
  - Proceed systematically though the steps of software development to produce a system meeting those requirements

Meaning of *Managerial Control*

- Managerial control means we are able to make rational choices about *development resources*
- Real projects have finite set of resources: time, people, money
- Must choose where, when, and how much resources are allocated
- Being in control means we can:
  - Decide in advance the level of resources needed to deliver software meeting requirements
  - Deliver that software on time and within budget

Analogy to Driving a Car

- In control: decide in advance where we want to go, how long, it should take, how much petrol it will take
- Out of control: end up at a different destination, takes twice as long as expected, uses twice as much petrol, etc.
- In control: decide in advance what capabilities and properties the software will have, how long it will take, how much effort
- Out of control: software is delivered with less or wrong capabilities, delivered late, over budget
- Many software projects are out of control in this sense
  - Perfect control is not possible
  - Requires constant feedback and correction
Application to DSD

- What does it mean to control a distributed development?
- Communication difficulties and context differences lead to coordination and control problems
  - Difficult managing development resources (schedule, personnel, budget)
  - Difficulty establishing requirements (eliciting, understanding, negotiating)
  - Difficulty effectively distributing work
  - Difficulty detecting and correcting conflicting assumptions
  - Difficulty managing change (especially requirements)

Relevant SE Areas

- Approach: apply software engineering processes, methods, and tools to mitigate risks
- In this course, focus on a few key areas
  - Process: How should we managing distributed resources?
  - Requirements: How do we ensure everyone is building the right system?
  - Software Architecture: How do we design for distributed development?
    - Divide the work, manage dependencies, reduce communication,
    - Quality assurance: How do we check our control?
- Other areas beyond scope of this course
  - Configuration management
  - Development environments
  - Etc.

Process Risks

- Process
  - Specified the set of activities, who performs the activities, when they are performed, and the artifacts produced
  - i.e., answers the questions ‘Who does what and when?’
- Example: a classic ‘waterfall’ process
  - What are the risks for a distributed development?
Process Goals

- Provide a common process for distributed teams
  - Provides the framework for coordination and collaboration
  - Defines which work products will be produced in what order
  - Defines who should do the work
  - Specifies what kinds of feedback are provided and when

- Goal: provide a process that addresses the problems of communication and coordination
  - Provides roles and mechanisms supporting explicit communication
  - Provides feedback to check how well communication is working
  - Avoids “big bang” integration

Requirements Risks and Goals

- Requirements risks in DSD
  - Many different stakeholders with different goals and different understanding
  - Risk of building the wrong system
  - Risk that cannot manage change effectively

- Requirements goals
  - Negotiate a common set of requirements among distributed stakeholders
  - Communicate requirements to all the developers so there is a common view of what should be built
  - Plan for change throughout development
    - Processes in place to manage change
    - Design for ease of change

Design Risks and Goals

- Relationship between system design and communication overhead
  - Must decompose the system into work assignments for distributed teams
  - System components depend on one another
  - Risk: More dependencies require more need for communication and coordination
    - i.e., Coupling == communication
- Software architecture goals
  - Decompose the system into work assignments that are as independent as possible
  - Work can proceed concurrently and independently
  - Little need for communication
  - Little need for one team to wait for another

- How do we design the software architecture to have these properties?
How should components be distributed among teams?

Goal: distribution of work on components requiring least inter-team communication

Control Risks and Goals

- Risk: loss of control of development
  - Development is not in control in some aspect
  - Poor visibility makes problem difficult to detect
  - Poor communication makes it difficult to correct
  - Changes disrupt control
  - E.g., distributed team is on schedule but misunderstands the requirements or has different interpretation

- Goal: detect and correct problems as soon as possible
  - Dynamic control of real processes requires feedback
  - Role of quality assurance processes (testing, review)
  - Need methods that work in spite of poor communication

Some SE Topics Covered

- Software processes and project planning
  - Which life cycle model to use
  - What kinds of activities and roles support distributed development
  - How to allocate work to distributed teams

- Software architecture
  - How to use architecture to manage dependencies
  - Which design principles and methods support concurrent development
  - Role of interfaces as contracts

- Software requirements
  - How to ensure mutual understanding of expected behavior and system qualities

- Software verification and validation
  - The role of feedback in maintaining control
  - Effective reviews
  - Distributed testing
Summary

- DSD presents certain specific problems that make it difficult to control software development
  - Create the desired system
  - Maintain budget and schedule
- Purpose of software engineering is to provide technology and techniques for maintaining control
- We will cover some specific techniques that address key DSD problems
  - Only a subset of useful methodologies

The Development Environment

- Effects of the development environment
  - Incompatible development infrastructures can result in inconsistencies between collaborating teams
  - Common environment can support communication and coordination
- Our exercise will use a common environment (assembla)
  - Provides common development workspace
  - Tools for synchronous and asynchronous communication
    - Email
    - IM
    - Wiki, document management
  - Some scheduling
  - Source control
- Demo
assembla Exercise

- Team exercises
- Each sub-team
  - Log on to your assembla workspace
  - Create a wiki page for team profiles
  - Add information for each team member
    - Name
    - Country and university
    - Programming languages
    - Favorite activities (sports, games, reading, movies, etc.)
    - Personal: other interests or anything about yourself you want to share
- Use the messages feature to agree on a team name.
  Post on the home page of the wiki

Questions?