Process Definition

Software Engineering
Computer Science 520/620
Spring 2014
Prof. Leon Osterweil

How to go about building it?
• High-level, large scale questions
• Low-level detailed questions
• Processes
  – High-level
  – Low-level

Complements prior focus on software artifacts:
This is a focus on software development activities

Simple example of a high-level process:
The Waterfall Model

This version suggests a much more complicated process

Which still leaves Key Questions Unanswered

Where does output go?
What to do when reviews fail?
Abstract Spiral Model: Suggests whole families of high-level processes

Reusing Based Development (e.g. Software Factory)

Cloud-Based Development

The Rational Unified Process

Many new software process ideas

Agile Methods

- Some add many details to abstract spiral model
- Some reject “waterfall-based” approaches
  - Too “heavyweight”
  - Is that exact sequence of steps always necessary?
  - Need for agility
- The rise of “agile methods”, “extreme programming”...
Some Extreme Programming (XP) Examples

- Test-first programming
- Pair programming
- Scrum

Test-First Programming

- “Test” the “program” before writing the code
- Boils down to:
  - Thinking about testing before coding
  - Doing analyses of pre-code artifacts as much as possible
  - Planning for testing/analysis right from the start
- Our course philosophy is very much in line with this

Pair Programming

- Code (and other artifacts as well) is produced by teams of two
  - “Driver”, who actually produces the code
  - “Navigator”, who watches, makes suggestions, spots defects, etc.
- Much research suggests that higher quality is obtained, and at costs that are comparable to single-programmer approaches

The Scrum: No Sequential Phases

- Software development in a sequence of “sprints”
  - Usually 30 sprints
- Each sprint lasts a day
- Sprint starts with a short meeting
  - Every team member has 2-3 minutes
- Scrum starts with overall goal-setting
  - A “burndown list”
- Scrum ends with evaluation
  - And planning for next scrum
- Main goals
  - Empower the team
  - “Time boxing” to keep things from taking too long
  - Risk mitigation

Scrum in Practice

- It is very popular
  - Just about everyone is “doing it”
- Lack of clear specification makes it hard to know for sure
  - Who is really doing it
  - What they are really doing
  - What actually works, and what doesn’t
- Meets some goals very well
  - Teams tend to feel empowered
  - Time boxing limits unexpected overruns

Who Should Use Scrum—and who should not

- Works better for smaller teams
  - Up to 8-10
- Works better when teams are geographically, physically close
- Works better for smaller, less complex, software projects
  - Where work can be broken up into smaller pieces better
Need a focus on process in order to complement our previous focus on product components

Being Precise About Processes

- Processes are REAL entities
- Important to define them
  - Completely
  - Clearly
  - Precisely
- For all relevant stakeholders
  - Developers
  - Customers
  - Managers
  - Regulators
  - Etc.

Processes as Software

- Consist of:
  - Process Requirements, the basis for
    - Process design, evaluation and improvement
  - Process Specification/Modeling/Design
    - Support for conceptualization, visualization
    - Process Code
      - Provides rigor and complete details
      - For for execution and tool integration
    - Process Analysis, Measurement, and Evaluation
      - Basis for....
  - Process Maintenance (Improvement)
- Develop processes using a process development process

Summary

- Software products are
  - Large, complex, tightly interconnected
  - Built by processes
- Software processes are
  - Products too
  - Processes and Products each contain the other
  - Processes and Products are built out of the same sort of material

Representations of Software Development Processes

- We have just seen a few attempts
  - DFGs
  - CFGs
  - UML
  - Combinations
- Could have seen FSMs, will see Petri Nets.
- Software processes are very complex, though
  - Require a great deal of modeling semantics

As we define software products as instances of types, we will also define the processes by which they are developed and related to each other by defining the processes for doing these things
Some Processes May be too complex for pictures

- What is happening when we do “rework”?  
  - Return to a previous point in the process  
  - But using different artifacts  
- Reworking is not looping  
  - It is more like recursion  
- Easy pictures (e.g. DFGs) don’t represent this well

Process Representation

- Who are the stakeholder groups for process representations?  
  - Developers  
  - Managers  
  - Customers  
  - Testers  
  - Etc.  
- What representation notation?  
  - We have already seen:  
    - DFGs, CFGs, Message Sequence Charts

Petri Net for a low-level requirements specification process

A low-level design subprocess

Can also use Well-Defined languages

- Diagrams support clarity, good for customers, ??  
  - Pictures support intuitive reasoning  
  - Help identify gaps, shortcomings, weaknesses  
  - Suggest truths, theorems, facts  
  - But are generally based upon very weak semantics  
    - Lack breadth of semantics  
    - Often lack precision and detail  
- Formal Languages, good for developers, ??  
  - Strength is precision and rigor  
  - Broad semantics are possible  
  - Often feature considerable detail (that may interfere with clarity)
Programming Languages

- Procedural
- Rule-Based
- Functional
- Combinations of the above
- Etc...

HFSP design model

HFSP: A Functional Decomposition Language

More Elaboration

The Little-JIL Process Language

- Vehicle for exploring language abstractions for
  - Reasoning (rigorously defined)
  - Automation (execution semantics)
  - Understandability (visual)
- Supported by
  - Visual-JIL graphical editor
  - Juliette interpreter
The “Step” is the central Little-JIL abstraction

Process definition languages are hard: Must address many issues
- Blending proactive and reactive control
- Coordinating human and automated agents
  - Without favoring either
- Dealing with exceptions
- Specification of resources
- Concurrency
- Real time specification
- Assignment of agents
- Scaling
- Reuse (e.g. through abstraction)
- Preemption/abortion

Four parts to a Little-JIL Process
- Coordination diagram
- Artifact space
- Resource repository
- Agents

Hierarchy, Scoping, and Abstraction in Little-JIL
- Process definition is a hierarchical decomposition
- Think of steps as procedure invocations
  - They define scopes
  - Copy and restore argument semantics
- Encourages use of abstraction
  - Eg. process fragment reuse
Proactive Flow Specified by four Sequencing Kinds

- Sequential
  - In order, left to right
- Parallel
  - Any order (or parallel)
- Choice
  - Choose from Agenda
  - Only one choice allowed
- Try
  - In order, left to right

These step kinds support human flexibility in process performance

Iteration usually through recursion
Alternation using pre/post requisites

Pre- and Post-requisites

- Steps guarded by (optional) pre- and post-requisites
- Are steps themselves
- Can throw exceptions
- May be executed by different agents
  - From each other
  - From the main step

Exception Handling: A Special Focus of Little-JIL

- Steps may have one or more exception handlers
- Handlers are steps themselves
  - With parameter flow
- React to exceptions thrown in descendant steps
  - By Pre- or Post-requisites
  - Or by Agents

Four different continuations on exception handlers

- Complete
  - Handler was a “fixup”; substep is completed
- Continue
  - Handler cleaned up; parent step is completed
- Restart
  - Handler cleaned up; repeat substep (deprecated)
- Rethrow
  - Rethrow to parent step
This is actually an ADL (or a SOA Composition Language)

- Steps represent components (some may be services)
- They also represent the semantics of complex connectors
- They specify which components are services
  - And what kinds
- More details...

Artifact flow

- Primarily along parent-child edges
  - As procedure invocation parameters
  - Passed to exception handlers too
  - Often omitted from coordination diagrams to reduce visual clutter
- This has been shown to be inadequate
  - Artifacts also need to flow laterally
  - And subtasks need to communicate with each other

Channels and Lateral flow

- Channel is like a queue in some ways
- Can specify step(s) that can add artifacts
- And steps that can take them
- All artifacts must be of the same type
- Generalizations are needed
Resources
- Entities needed in order to perform step
- Step specifies resource needed as a type
  - Perhaps with attributes, qualifiers
- Resource instances bound at runtime
- Exception when "resource unavailable"

Examples of Resources
- Access to artifacts: shared document, locks on databases
- People: various kinds with varying skills
- Tools: compilers, CASE tools
- Agents: Each step has a distinctly identified unique resource responsible for execution of the step (and all of its substeps)

May be complex relations among them

Resource Request Example
Agent: OODDesigner; expert tool: ClassDiagramEditor artifact: DiagramReposLock

Resource request is a query on the Resource specification repository

Agents
- Collection of all entities that can perform a step
  - Human or automated
- Process definition is orthogonal to assignments of agents to steps
  - Path to automation of process
- Have freedom to execute leaf steps in any way they want

Try and Choice Step Kinds support human (agent) flexibility

Implement
- Reuse_Implementation
- Custom_Implementation

Look_for_Inheritance
- Look_for_Parameterized_Class
- Look_for_Objects_to_Delegate_to

Timing
- Step has (optional) deadline specification
- Exception when deadline exceeded
- Parent can proceed
  - Child may be unaware of this
Preemption Semantics

- Need to allow one step to terminate execution of another step
  - Terminated step must allow this
- Some variants of this
  - Abort a step
  - Suspend a step
  - Rollback, compensate, etc.
- Only abort is implemented now
A better basis for proceeding

How to do this?

And how to do these too?

Example
Requirement Specification Process

Develop Rqmt Element

Declare and Define Rqmt

Declare Rqmt Element

Define Rqmt Element

Requirements

Inter-requirement Consistency Check

Being Precise about Requirements Processes

- Helps developers understand
  - Other stakeholders too
- Basis for automated support
- Being precise about processes is closely tied to being precise about artifacts
  - More on this shortly
Requirement Processes

- Elicitation
  - How to ascertain requirements
  - Interviewing, classifying, organizing
  - Emphasis on perspectives/viewpoints
- Review
  - How to determine consistency, completeness, etc.
  - Emphasis on analysis
  - Need for semantic basis and inference reasoning
- Revision/improvement/enhancement
  - How to add, delete, modify
  - Rereview: how to determine consistency of modified requirement specification

Example Requirement Specification Process

Elaboration of Define Rqmt Element

Better Elaboration of Define Rqmt Element

Focus on Evaluation

Focus on Evaluation

Helps if fcnl rqmt is defined as a DFG
Examples of what to check and how

Internal Consistency:
\[ \forall r \in R, s \in \text{children}(r) \Rightarrow \text{parent}(s) = r \]
\[ \forall r \in R, t \in \mathbb{R}, \text{testresult}, \text{pass}(\text{testresult}) = \text{True} \Rightarrow \text{pass}(t) = \text{True} \]
\[ \forall s \in \text{descendant}(r) \]

Interartifact Consistency:
\[ \forall r \in R, i \in \text{inputs}(r) \Rightarrow i \text{ is an input to the node} \]
\[ n \text{ in the DFD that defines the functionality of } r \]

Examples of what to check and how

Internal Consistency:
\[ \forall r \in R, s \in \text{children}(r) \Rightarrow \text{parent}(s) = r \]
\[ \forall r \in R, t \in \mathbb{R}, \text{testresult}, \text{pass}(\text{testresult}) = \text{True} \Rightarrow \text{pass}(t) = \text{True} \]
\[ \forall s \in \text{descendant}(r) \]

Interartifact Consistency:
\[ \forall r \in R, i \in \text{inputs}(r) \Rightarrow i \text{ is an input to the node} \]
\[ n \text{ in the DFD that defines the functionality of } r \]

Many Details Left Out
Many Other Alternatives

- When to check what
- How to do the checks
- How to respond
  - And when
- Etc.
- Being more specific about process entails being more specific about artifacts/products

Focus on Rework/Evolution
Requirements Rework May Be Triggered During Design

Requirements Rework Process

Contains a Previously Executed Step

That We Saw Previously Here

Requirements Rework

Requirements Rework

Same exception thrown

Invocation of step originally defined as substep of Requirements

Invocation of step originally defined as substep of Requirements
Requirements Rework

Data Flow Definitions

Analogous View of Design

Coding Too

Scrum: Being Precise about a Process

Scrum in Practice

- Software development in a sequence of "sprints"
  - Usually 30 sprints
  - Each sprint lasts a day
  - Sprint starts with a short meeting
    - Every team member has 2-3 minutes
  - Scrum starts with overall goal-setting
    - A "burndown list"
  - Scrum ends with evaluation
    - And planning for next scrum
  - Main goals
    - Empower the team
    - "Time boxing" to keep things from taking too long
- Risk mitigation
Who Should Use Scrum—and who should not

• Works better for smaller teams
  – Up to 8-10
• Works better when teams are geographically, physically close
• Works better for smaller, less complex, software projects
  – Where work can be broken up into smaller pieces better

Suggests that the details matter

• How many sprints?
• What policies for managing the burndown list
• Etc.
• Different variants may be more suitable for different application domains
Scrum: Deadline Specification

Development Iteration

Sprint Planning Meeting

Sprint

Sprint Review

Sprint Retrospective

product: Product

sprint backlog channel: Backlog Channel

sprint backlog

sprint backlog

agent: ScrumMaster

owner: ProductOwner

deadline: Hours = 4

product: Product

agent: team

Now Elaborate on the Sprint Step

Sprint: Activity Skeleton

Daily Sprint

Daily Scrum

Revise Sprint Backlog

= X X 30 + *

product: Product

product: Product

Checked Work

Sprint: Artifact Flow

Sprint: Channel Communication

Daily Scrum

Sprint Planning Meeting

Sprint

Sprint Review

Sprint Retrospective

agent: ScrumMaster

owner: ProductOwner

deadline: Hours = 4

product: Product

agent: team
### Scrum: Being Precise about a Process

- Software development in a sequence of "sprints"
  - Usually 30 sprints
- Each sprint lasts a day
- Sprint starts with a short meeting
  - Every team member has 2-3 minutes
- Scrum starts with overall goal-setting
  - A "burndown list"
- Scrum ends with evaluation
  - And planning for next scrum
- Main goals
  - Empower the team
  - "Time boxing" to keep things from taking too long
  - Risk mitigation

### Main Goals

- Empower the team
  - Who decides what is on the burndown list?
  - Who assigns tasks each day?
- Time boxing
  - How long between reports from team members?
- Risk mitigation
  - How long between system integrations?
  - How to assure competent people do most important tasks?
- Other goals?

### Our Approach

- What is the goal/role of each component type?
- What is the nature of it?
  - Eg. what internal structure does it have?
- What sorts of stakeholders are interested in it?
- What sorts of questions do they generally have about it?
- What sorts of relations must it participate in?
  - Internal relations
  - External relations
- What sorts of processes deal with it?

### Some Observations

- Process engineering is important, feasible
- Effective process languages are possible
  - Borrowing from programming languages helps
    - Abstraction, scoping, exception management, concurrency, etc.
  - Transactions and Real-time are needed too
- Analysis is feasible for detecting defects
  - Basis for systematic process improvement
- Process guided execution has value
  - Needs process guided user interface management

### Types of Software Product Components

- Specification of customer/user needs/desires
  - REQUIREMENTS
- Specification of potential solution or solution approach
  - ARCHITECTURE
- Reduction of solution approach to practice
  - DESIGN
- Solution
  - IMPLEMENTATION
- Evaluation of solution
  - TEST PLAN
  - ANALYSIS/TEST RESULTS
Types of Software Product Components

- Specification of customer/user needs/desires
  REQUIREMENTS
- Specification of potential solution or solution approach
  ARCHITECTURE
- Reduction of solution approach to practice
  DESIGN
- Solution
  IMPLEMENTATION
- Evaluation of solution
  TEST PLAN
  ANALYSIS/TEST RESULTS

Well-Integrated, consistent, correctly related to each other