What are these circles in this Software Product?

Products in other disciplines

- Homebuilding
  - Architect sketches
  - Blueprints
  - Appliance Manuals
  - Violation reports
  - ??
- Legislation
  - Hearings
  - Laws
  - Bureaucracies
  - Court cases
  - ??
Some Key Features of These Products

- Problem enunciation, understanding
  - What is the problem to be solved?
- Solution formulation
  - How might the problem be solved?
- Solution reduction to practice
  - How will the problem actually be solved?
- Solution implementation
  - The actual solution to the problem

Some Key Features of These Products

- Problem enunciation, understanding
  - What is the problem to be solved?
- Solution formulation
  - How might the problem be solved?
- Solution reduction to practice
  - How will the problem actually be solved?
- Solution implementation
  - The actual solution to the problem
  - Interconnections among all of these
  - Evidence of consistency
  - Intuition (?) about what makes them "good"
  - Schedules and development histories

Some Key Features of These Products

- Problem enunciation, understanding (requirements)
  - What is the problem to be solved?
- Solution formulation (architecture)
  - How might the problem be solved?
- Solution reduction to practice (design)
  - How will the problem actually be solved?
- Solution implementation (coding)
  - The actual solution to the problem
  - Interconnections among all of these
  - Evidence of consistency (analysis/testing)
  - Intuition (?) about what makes them "good"
  - Schedules and histories (development process)

Software Artifacts as Instances of Types

- Example: A requirement specification
  - A structure of instances of types
  - Probably hierarchical
  - Different types for different kinds of requirements
- Functional Requirement
  - A DFG?
  - An FSA?
- Timing requirement
  - A first order logic specification?
  - A Petri Net?
- Robustness requirement
  - An FSA?

How to represent all of this?

How to decide which to use?
Most of the rest of the course will examine the various key types of software product components and try to define them as (software) types.

Relations

- Many software artifacts are relations
  - DFG, CFG
- They define, correctness, “well-formedness”
- Are the basis for evaluation
  - Are all of the instances of a relation properly related to each other?
- Are used in evolution
  - If not, then remedying improper relations becomes a goal of evolution

Some Examples of “Relations”

- Executing this code must meet this requirement
- This code must conform to that design element
- This compiled code came from this compiler
- This design element addresses those requirements

Some Examples of “Relations”

- Executing this code must meet this requirement
- This code must conform to that design element
- This compiled code came from this compiler
- This design element addresses those requirements
- These lower level requirements are elaborations of these higher level requirements
- This is the date by which that test must be passed
- Component invocations conform to component abstract interface specifications
- Documentation describes the actual system
- ETC.....

Much of the rest of the course will examine the ways in which consistency is defined and determined through the use of relation specifications.
Much of the rest of the course will also be focused on the ways in which these artifacts are made and interconnected.

If the product is this complicated, then it probably takes a complicated process to make it.

How to define the process too?

Problems with the Waterfall

- It trivializes the process
  - No loops
  - No decomposition
- And things do not necessarily have to happen in this order either

OK, so here are some loops: Rework between phases

Still at Least One More Problem: Need both Control and Data Flow

- Which did we just see?
  - Could have been either (both?)
- DFGs lack control flow information
- CFGs lack data flow information
- When to do what?
- What really follows what, and when?
- Etc.
So how about this version?

Leaves Key Questions Unanswered

Requirements

Low-Level Design

High-Level Design

Data Flow

Control Flow

Code

Test

Where does output go?

What to do when reviews fail?

What portion of This activity should be done?

How do we break this cycle?

Iteration in process

- Loops somehow seem inherent in software development
- Waterfall Diagram CFG showed loops with no useful semantics
  - Clearly inadequate
- Putting in loops made things complicated
  - And less clear

What’s the Problem with Loops?

- Loops are hard to control
- What changes on each iteration
  - And what does not
- When to stop looping
  - And how?
- Familiar problems in application software
- Also present in software processes

What’s the Problem with Loops?

Question: What are the loops for?

The Shewhart/Deming Cycle:
W. Edwards Deming
- Father of modern manufacturing quality
  - Bell Labs in 1940s
  - Appreciated first by Japan
  - Now universally appreciated
- Popularized "Plan-Do-Check-Act"
- Credits PDCA to Walter Shewhart
  - Just the "scientific method"? (Francis Bacon in 17th Century)

The Shewhart/Deming Cycle: Each Iteration Yields Improvement

Looping in Software Development Should Be for Risk Reduction (?)
- What are the risks?
  - Failing to satisfy stakeholders
  - Who are the stakeholders, what are their stakes? What are the costs of failing a stakeholder?
  - What are the costs of being sure you don’t fail?
- Waterfall does not suggest or support this idea
- More contemporary development approaches focus on this issue better
- E.g. Boehm’s Spiral Model makes these issues clearer

Some More Contemporary Software Development Approaches
- Boehm’s Spiral Model
- Kruchten’s RUP (Rational Unified Process) Model
- Agile Models
  - e.g. Scrum
- Flexible Models
  - e.g. Process Programming

Boehm’s Spiral Model
- Risk Analysis
- Requirements
- Design
- Code
- Test
- Review
- Design
- Code
- Test
- Review
- Plan next phase
- Develop & verify
Abstract Spiral Model

Different Traversals of the Spiral Model to address different risks

And Yield Some New Approaches and Some Old Ones

The Waterfall

More Realistic Waterfall

Reuse Based Development
"Throwaway" prototyping

Create a history of past use as part of the process to get a firm grip in risk issues in order to control them better.

Evolutionary prototyping

Similar to previous. Difference is the Scale of the prototype

The Rational Unified Process

Use UML to define the process. This is a Message Sequence Diagram

The Rational Unified Process

Use UML to define the process. This is a Message Sequence Diagram

Many new software process ideas

- 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
- Etc.

More on these later
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

Representations of Software Development Processes

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

Being Precise About Processes

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

Need a focus on process in order to complement our coming focus on product components
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 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

Process Representation

- Who are the stakeholder groups for process representations?
  - Developers
  - Managers
  - Customers
- For Microprocess?
  - Developers and Managers
- What representation notation?
  - Need more details and rigor
  - Can be more technical, formal
- Diagrams could be OK for customers
  - Would need to be consistent with formal notations