

ICS121: Software Tools and Methods

Fall Quarter, 2005

Dan Frost

based on slides by Prof. Sims

What is this course about?

- You know:
 - Concerns of software engineering
 - Software process models
 - Problems that arise in software engineering
- You will learn in this class:
 - Professional means to solve those problems
 - Tools and methods used in industry
 - But no programming project
- For example:
 - UML
 - Eclipse
 - Cost estimation

Objectives for the Course

- Increase your exposure to a number of software tools
 - Eclipse, CVS, Ant, Junit, Rational Rose
- Give you experience working with methods
 - UML, project management
 - Ideas more important the particular tool.
- Hands on
- Interactive
- Lots of work, but you'll learn useful skills.
 - All of these tools and methods (or ones similar to them) are used in industry.
 - Good for your resume

Topics for the quarter

1. Software Process and Technology
2. Quality Assurance
3. UML for Specification and Design

Textbooks

- van Vliet, Pressman
 - 52 text
 - Background
 - Motivation
- Scott
 - UML
- Brooks
 - Essays
- Assorted Papers
 - Fishman
 - Humphrey
 - Miller and Collins
- Gallardo
 - Guide to Eclipse
- Syllabus contains assigned readings by topic
 - May not discuss all readings in class
 - Readings *will* be on the examinations

Grading

- Assignments 40%
- Laboratories 14% (7 out of 8)
- Midterm Exam 16%
- Final Exam 30%

Handing In Assignments and Labs

- Electronic submissions through Checkmate
- Strict lateness policy because one of the goals of software engineering is to deliver projects on time, on budget, and with fulfilled requirements. (See Brooks, Ch 8)
 - Deduct 1% per hour late
 - Penalties for incomplete assignments
 - Lateness is calculated from date and time that assignment is due
 - Checkmate time stamps the entire assignment with the time the latest file is submitted

Cheating

- Letter in your UCI file
- Course grade: F
- No team work on assignments
- Discussing an assignment is OK, copying the solution is not
- Anything copied from books or Web pages must be quoted and the source must be given
- Does instructor have access to assignments from previous ICS 121s?

Are you going to curve the grades?

- No
- “Curving” means transforming the grades so they fit a normal distribution
 - Can be beneficial for below average grades, but can also be detrimental for above average grades
- You will be graded on your achievement
 - All assignments (and tests) have solutions and can be completed if you start ahead of time
 - It is possible for the entire class to earn A’s (if you work)
- Standard straight scale: 90s are As, 80s are Bs, 70s are Cs, 60s are Ds, you get the picture.
 - I reserve the right to make small adjustments.

Why do we need tools and methods?

- As teams and projects become larger, practices that worked with 1 or 2 people don't work any more
- Most students:
 - Use a text editor (TextPad), a few use an IDE
 - Use the file system to manage projects
 - Compilation done by hand or scripts
 - Print statements used for debugging
 - No install support
- **Problem: This approach doesn't scale up**
 - More people, bigger code, different versions, multiple platforms (development and delivery)

Example: Longhorn Project

- 5000 developers (excludes non-tech staff)
- 40 million lines of code
 - 16 million added in the last three years
- Daily builds and regression testing
 - Takes 3 working days from the time you submit changes until you receive an executable back
- ~1.7 testers to every programmer
- Needs to be backwards compatible
- Installation needs to work on millions of machines

Challenges to Microsoft

- Logistics

- How do you design a process that will allow 5000 people to work together at the same time?
- How do you test so much code? For so many platforms?

- Design

- How do you do design on a system with 40 million lines of code?
- How do you maintain conceptual (architectural) integrity?

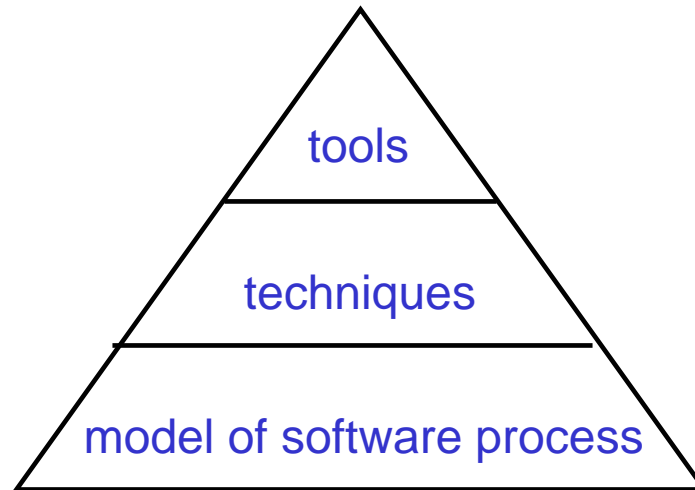
Software Technology

- Types of software technology
 - Tools
 - Methods
 - Notations
 - Process Models
- How do they help?
 - Automate tasks or help people do complex tasks
 - Improve software quality
 - Increase productivity
 - Permit verification and conformance checking
 - Project tracking
 - Establish procedures for doing things

Software Tools

- A software tool supports a specific task in the software development process
 - Example: a compiler, model checker
- An environment supports an entire development process
- A workbench is a set of tools with a limited scope
 - Note: “Eclipse Workbench” does not use the term in this sense

Selecting a Tool



- Ideal: start decision making from the bottom
- In practice: start from the top
 - “software lemmingengineering”

Methods

- A method is a technical prescription for how to perform a collection of activities, focusing on integration of techniques and guidance on their use.
 - Example: Rational Unified Process
- A technique is a prescription of how to perform a particular activity
 - May include rules on how to describe a product of that activity in a particular notation.
 - Smaller than method
 - Example: unit testing

Notations

- A notation is a representation scheme (or language) for expressing things
 - Examples: first order logic, UML, algebra, Roman numerals
- A process model is an abstract description of how to conduct a collection of activities, focusing on resource usage and dependencies between activities.
 - Often expressed using a notation
 - Example: Waterfall
- A process is an enactment of a process model, describing the behavior of one or more agents and their management of resources

Concept Check

- Classes of Software Technology
- Tool
- Method
- Technique
- Notation
- Process Model
- Instances
- TextPad
- Compiler
- Black box testing
- UML
- Spiral model
- RUP
- Object-oriented analysis and design
- RAD (Rapid Application Development)

Review of Software Process Models

- Waterfall
- Incremental
- Spiral
- Evolutionary

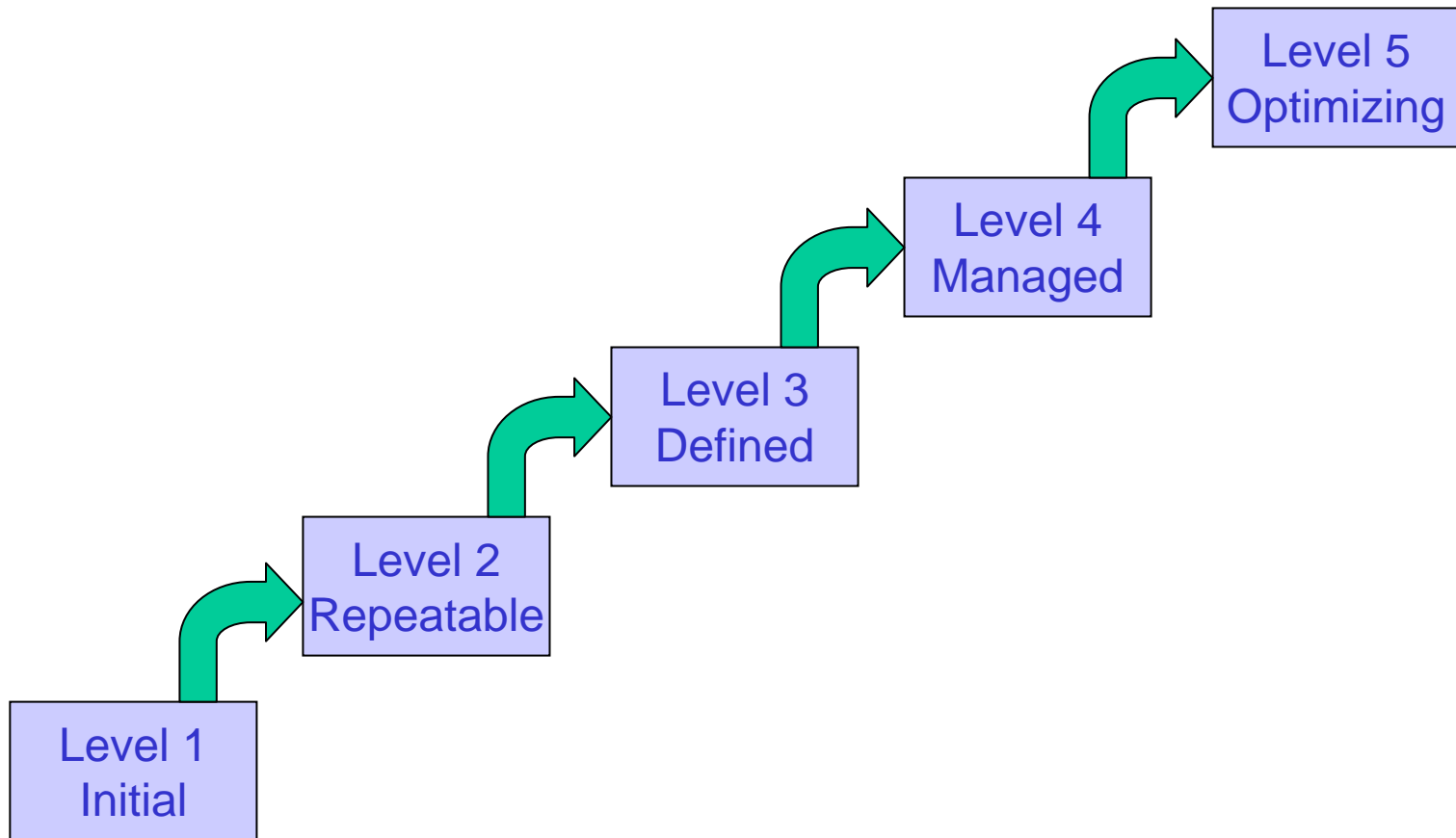
More Software Processes

- Capability Maturity Model (CMM)
 - Technically, a process improvement model
- Extreme Programming (XP)
- Synch-and-stabilize
 - Created at Microsoft
- Rational Unified Process (RUP)

Capability Maturity Model

- Created by members of SEI (Software Engineering Institute) during 1980–90s
 - SEI is an research institute at Carnegie Mellon University, funded by DOD and others
- Studied many projects to find practices that worked and organized the knowledge into a prescription
- Idea: what you *do* achieve is limited by what you *can* achieve
 - Upper limit on performance is potential
 - Potential = capability maturity

CMM Levels



CMM Levels

- Characterized by description and key areas or technologies
- To enter next level, ALL processes must be at that level

CMM Levels

Level 1 Initial

- No effective management procedures or project plans
- No organizational mechanisms to ensure that project control is used consistently
- Success is unpredictable

Level 2 Repeatable

- Formal management, quality assurance, and configuration management are in place
- Can repeat previous successes
- No documented process model, depends on talented individuals

CMM Levels

Level 3 Defined

- Documented process and quantitative process improvement in place
- Mechanisms to enforce process are used consistently

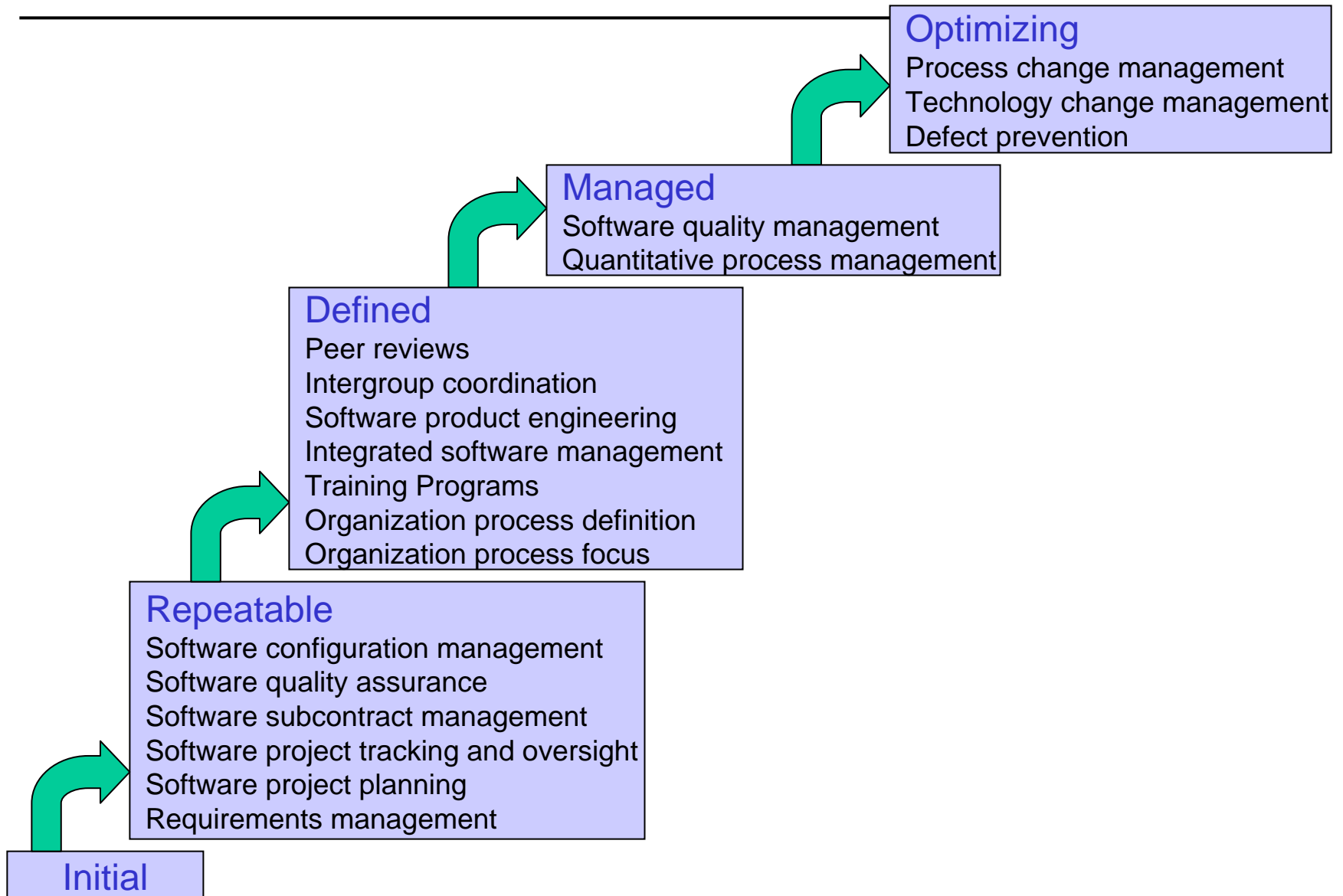
Level 4 Managed

- Process and product metrics are collected and fed back into process improvement

Level 5 Optimizing

- Committed to continuous process improvement
- Budgets and project plans include resources to support process improvement

Key Process Areas



Level 2 Key Process Areas

- Requirements management
 - Establish common understanding with customer for project goals
- Software project planning
 - Plans with realistic estimates
- Software project tracking and oversight
 - Establish visibility into project progress
- Software subcontract management
 - Ability to select qualified subcontractors and manage them effectively
- Software quality assurance
- Software configuration management

Level 3 Key Process Areas

- Organization process focus
 - Allocate responsibility for process improvement
- Organization process definition
 - “software assets” and meaningful data
- Training Programs
 - Improve skills and knowledge of team members to increase effectiveness
- Integrated software management
 - Process that integrates management with engineering
- Software product engineering
 - Consistent engineering process using well-described methods

Level 3 Key Process Areas

- Intergroup coordination
 - Establish means for team to participate in projects with other teams
- Peer reviews

Level 4 Key Process Areas

- Quantitative process management
 - Managing using numbers
 - Know when there are exceptions in the time taken, resources used, and frequency of activities
- Software quality management
 - Quantitative understanding of quality goals for software products and organizational performance

Level 5 Key Process Areas

- Defect prevention
 - Identify root causes of defects and prevent recurrence
- Technology change management
 - Orderly transfer or injection of beneficial software tools and methods
- Process change management
 - Continually improve the software processes in the organization
 - Improve software quality, increasing productivity, decreasing cycle time

Costs and Benefits of CMM

- Takes about 2 years per level to move from initial to defined
- Costs about \$500–2000 per employee per year
- Some have reported a 5:1 ROI
 - Every \$1 invested has resulted in a \$5 savings

Some Critical Notes

- Better suited to large organizations
 - Cost and overhead
- Model focuses on project management rather than product development
- Capability assessment too strict and too coarse-grained
 - A Level 5 organization can still have a Level 1 project
 - A single number is like an ‘instrument panel’ with one gauge

On the other hand...

- Level 5 organizations produce high quality software
- Large projects need this level of control
 - 4 vs. 400 vs. 4000 developers
 - Driving an aircraft carrier vs. driving a bicycle
- Some applications require a high level of confidence
 - Defense, aerospace, transportation control, medical devices

A CMM Level 5 Organization

- Space Shuttle Onboard Flight Software
 - See Fishman reading
- As of 1996, current version was 420,000 LOC
 - 1 error in each of last 3 versions
 - 11 errors total over 17 versions
 - Comparable commercial code would have 5000 errors
- How do they do it?

Space Shuttle Software Process

1. The product is only as good as the plan for the product.
2. The best teamwork is a healthy rivalry.
 - Coders vs. verifiers
 - Find 85% of errors before testing phase, 99.9% before delivery
3. The database is the software base.
 - Change history embedded into source code
 - Error base
4. Don't just fix the mistakes– fix whatever permitted the mistake in the first place.

Shuttle Software

- Highly managed, disciplined software development
 - Few surprises and non-chaotic
 - No flying by the seat of your pants
 - Relatively high number of women (12 out of 22 in one meeting)
- Expensive. As of 1996,
 - 260 people on team
 - \$35 million per year
 - 420 000 lines of code per version

Agile Methods

- Complaints: The CMM is too unwieldy. There are too many metrics and too much documentation. It gets in the way of what's really important—writing code!
- Response: agile methods
 - Agile means being able to move quickly
 - Mentally quick and resourceful
 - For smaller teams and small businesses
- Most famous (or infamous?) example is Extreme Programming

Extreme Programming (XP)

- Kent Beck invented XP by the seat of his pants in 1996 to fix a disastrous project at Chrysler
- “Extreme Programming Explained: Embrace Change” by Beck published in 1999
 - Current hot topic in software process
 - Loved and hated
 - Tries to associate with a software process with extreme sports
- Idea: Take a good programming practice and push it to the extreme.
 - Example: testing
 - Since testing is good, do it all the time, even before we have code.

Extreme Practices

- If code reviews are good, we'll review all the time (pair programming).
- If testing is good, everybody will test all the time (unit testing).
- If design is good, we'll make it part of everybody's daily business (refactoring).
- If simplicity is good, we'll always leave the system with the simplest design that supports the functionality (the simplest thing that could possibly work).
- If architecture is important, everybody will work defining and refining the architecture all the time (metaphor).
- If integration testing is important, then we'll integrate and test several times a day (continuous integration).
- If short iterations are good, we'll make the iterations really, really short– seconds and minutes and hours, not weeks and months and years (the Planning Game).

Premise for XP

- **Analogy to driving**
 - Small changes, small results, continuous feedback
- **Four Values**
 - Communication, simplicity, feedback, and courage
- **The Principles**
 - Concrete applications of the principles
 - Rapid feedback; assume simplicity; incremental change; embracing change; quality work
 - Teach learning; small initial investment; play to win; concrete experiments; open, honest communication; work with people's instincts, not against them; accepted responsibility; local adaptation; travel light; honest measurement
- **Four Basic Activities**
 - Coding, testing, listening, and designing

XP Concepts

- XP is a set of key practices that suggest a software development process.
- Key concept: Embrace change. Rather than avoid changes, try to reduce the cost of making changes.
- Key concept: Defer costs. Rather than face every problem up front, try to start with a small subset and incrementally plan and carry out improvements.

Some XP Terminology

- **User story**
 - A short, concrete paragraph about how a user might use one feature of the system.
- **Test case**
 - Programmatic unit test case. Each test case is very small and would take only an hour or less to write.
- **Iteration**
 - The work it takes to implement a few user stories. Usually 1–4 weeks.
- **Release**
 - Several iterations happen lead up to one release. The release produces an externally visible version of the software. A release can take a few months.

Twelve Key Practices of XP

Programmer Practices	Simple Design Test-driven development Refactoring Pair programming Collective code ownership Continuous integration Coding standards
Management Practices	Planning Game Small releases 40-hour week
Customer Practices	On-site customer Metaphor