CS 370
Overview of Software Engineering Approaches

DR. MICHAEL J. REALE
FALL 2015
We will now go over some of more well-known software engineering approaches/methods

- This list isn’t ANYWHERE NEAR exhaustive!
- Also, the landscape is always changing

Boehm-Turner sort these methods (approx.) from most agile to most plan-driven

- Also sorted by how much constraint it puts on the developers
- That said, some of the approaches can be more/less one or the other based on the project conditions
<table>
<thead>
<tr>
<th>Agility Rank</th>
<th>Method</th>
<th>Very Low</th>
<th>Low</th>
<th>Med.</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scrum</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Adaptive Software Development (ASD)</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Lean Development (LD)</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Crystal</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>eXtreme Programming (XP)</td>
<td>● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Dynamic Systems Development Method (DSDM)</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Rational Unified Process (RUP)</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Team Software Process (TSP)</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Feature-Driven Development (FDD)</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Capability Maturity Model Integration (CMMI)</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Capability Maturity Model for Software (SW-CMM)</td>
<td>● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Personal Software Process (PSP)</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Cleanroom</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scrum: Concept

- Scrum
  - Developed by Ken Schwaber and Jeff Sutherland
  - Concept: Software Development is...
    - NOT a defined process
    - An empirical process
      - Complex input/output transformations that MAY or MAY NOT be repeated under different circumstances

Scrum: Name Origins

- Name comes from a *scrum* in rugby:
  - Teams attempt to move against each other in large, brute-force groups
  - Must quickly:
    - Counter opponents’ movements
    - Exploit opponents’ weaknesses without luxury of planning
Scrum: Project Management

- Focus on project management

- **Scrum Master** = project/team management
  - Kind of like a consultant and/or coach as well

- **Daily Scrum meetings**
  - Less than 30 minutes
  - Allows team to monitor status and communicate problems
Scrum: Sprints and Features

• **Product Backlog**
  - Contains features/functions/tech enhancements planned for the project

• **Sprint** = 30-day development cycle
  - Before Sprint, two meetings:
    - What features do we want to implement in this Sprint?
    - Plan out the work itself
  - **Sprint Goal** = minimum success criterion for Sprint
    - Keeps team focused on big picture
Scrum Development Cycle

http://www.methodsandtools.com/archive/scrum1.gif

Sprint Backlog

Backlog tasks expanded by team

Daily Scrum Meeting

24 hours

30 days

Product Backlog

As prioritized by Product Owner

Potentially Shippable Product Increment

Source: Adapted from Agile Software Development with Scrum by Ken Schwaber and Mike Beedle.
Scrum Overview

- Can be used to introduce Agile methods to traditionally disciplined environment
  - Don’t have to overhaul how you do everything

- Can be scaled up to larger projects
  - Individual Scrum team coaches → part of higher echelon team of coaches that span several products
    - Maintain communication and prevent conflict among teams
  - Usually need Cockburn Level 3 coaches and personnel for this to work
Adaptive Software Development (ASD)

- From rapid application development work by Jim Highsmith and Sam Bayer
- Basic idea: continuous adaptation of the process to the work at hand is normal
- Acknowledges uncertainty and questions deterministic Waterfall model
  - “We can finally admit we don’t know everything.” – Jim Highsmith

http://jimhighsmith.com/
https://twitter.com/sambayer
ASD Life Cycle

- Instead of Waterfall model (or even iterative mini-Waterfalls), repeating series of cycles that:

  - **Speculate**
    - Planning is tenuous
    - Encourages exploration and experimentation

  - **Collaborate**
    - Complex apps → too much information flow for one person (or even one group) to keep track of
    - Must work together to produce results, share knowledge, and make decisions

  - **Learn**
    - Admitting humans are fallible → seek to improve ourselves and our work
    - Project retrospectives and customer focus groups
      - End of iteration (not end of project)

http://jimhighsmith.com/dont-plan-speculate/
ASD Life Cycle

- ASD cycle is:
  - **Mission-focused**
    - Requirements at start may be fuzzy → mission must be clear
    - Mission statements
      - Boundaries rather than fixed destination
      - Allow exploration at beginning → narrow in focus later
  - **Feature based**
    - Results, not tasks
    - Almost invariably → software feature = results
  - **Iterative**
  - **Timeboxed**
    - I.e., time is fixed, but you may relax WHAT you do in the time period
    - Focuses and forces trade-off decisions
  - **Risk driven**
  - **Change tolerant**
    - Leveraging change as a competitive advantage
ASD Overview

- Need Cockburn Level 3 guru to tailor ASD to specific project
  - Designed to be managed “balanced on the edge of chaos”
  - Little specifics on how things get done

- May be difficult to apply to large, critical, relatively stable development efforts
  - Method is non-prescriptive
Lean Development (LD)
Lean Development

- **Lean Development**
  - Proprietary approach by Bob Charette (and his company ITABHI)
  - Agility = tolerance to change
  - Three-tiered approach to competitiveness based on change
    - “Risk entrepreneurship” = ability to turn risk into opportunity
LD Process Lifecycle

- More traditional complete life cycle framework
- Three phases
  - Startup
    - Overall planning
      - Business cases and feasibility studies
  - Steady state
    - Design-build in series of short spirals
  - Transition-renewal
    - Product maintained in this phase
    - Documentation developed and delivered
- Product-focus + Product lines
  - Software can be reused across several models of same generic product
  - “Domain, not point solutions”
- **LD focuses on entire enterprise**
  - Involves EVERYONE in chain (from CEO down)

- More of a business strategy and project management approach
  - Not really development process
  - Nonspecific about particular software development practices, policies, and guidelines
  - Does stress small, co-located, multi-disciplinary teams
  - Supports concept of **humanware**
    - Empowering user through better hardware/software interfaces
    - “Those using lean development need to achieve satisfaction through their work.”
• First agile method to address top executives
  ○ Strategic, risk/opportunity-based approach resonates with management

• Most explicit focus on assessing and achieving business value
  ○ “...maximizing customer satisfaction through creating real business value at an acceptable level of technical quality.”
  ○ Deliver value sooner rather than later:
    ▪ 80% now is better than 100% tomorrow
  ○ Buy rather than build if you can

• Allows for other approaches/methods/processes:
  ○ “Lean development aims to increase revenue, reduce costs and increase profits by creating increased customer value. If there are better ways of doing so than lean development, use them.”

• Proprietary
  ○ Hard to implement (without their help...)

Proprietary
Crystal
Crystal

- Crystal
  - Developed by Alistair Cockburn
    - Outgrowth of his global consulting efforts
  - Not single method ➔ framework/family of methods
    - Vary approach based on environment and project
  - Use “crystal” as metaphor for each method:
    - “color”
    - “hardness”

Indiana Jones and the Crystal Metaphor

- "Color" = # of people involved
  - Clear 1-6
  - Yellow 7-20
  - Orange 21-40
  - Red 41-100

- "Hardness" = criticality in terms of type of loss
  - Comfort (C)
  - Discretionary money (D)
  - Essential money (E)
  - Life (L)
Crystal Values

- Crystal methods share two values:
  - People and communications centric
    - Emphasis on:
      - Reflective improvement
      - Close communication and co-located teams
      - Personal safety in communication (with boss or co-workers)
      - Easy access to expert users
      - Clear objectives and time/space to focus
      - Tools to aid testing/integration
  - Highly tolerant
    - “Ultra-light”
      - Least amount of paperwork/overhead/bureaucracy possible for your project
    - “Stretch-to-fit”
      - Start small/simple and grow method(s) to right size
    - Allows use of other methods (or even substitution of parts of Crystal method)
Crystal Comments

- Only two colors (Clear and Orange) fully elaborated
  - Strong approach where well defined
  - Difficult to apply if not Clear or Orange

- Can be tailored by Cockburn Level 2 person
  - Unlike ASD

- “Lighter is better as long as it lasts”
  - Implement harder version ONLY if significant consequences (or risks) involved
eXtreme Programming (XP)
eXtreme Programming (XP)

- **eXtreme Programming**
  - Most widely recognized Agile method
  - By Kent Beck, Ward Cunningham, and Ron Jeffries
    - Experience at Daimler Chrysler (Chrysler Comprehensive Compensation System or C3 project)
  - Often used in tandem with Scrum
**XP: Values**

- XP → based on four values and 12 practices (extended in various ways)

- Four XP values:
  - Communication
    - Poor communication → why most projects fail
    - Force communication in positive fashion
  - Simplicity
    - Develop simplest product that meets customer’s needs
  - Feedback
    - Obtain and value feedback from customer, system, and developers on team
  - Courage
    - Be prepared to make hard decisions that support other principles and practices
XP: Planning

- **Begin with planning game**
  - Customers and developers negotiate requirements
  - Requirements → “user stories”
  - Design → “metaphor”

- **Plan multiple iterations**
  - Decide requirements to implement for given iteration
  - Multiple small releases
“User stories” = requirements in XP

- From customer’s/user’s perspective
  - Should be written by customer
  - Should not be too technical
    - “Students can purchase parking passes online” \(\rightarrow\) good
    - “Transcripts will be available online via a standard browser” \(\rightarrow\) too technical

- Formally, has format:
  - “As a (role) I want (something) so that (benefit).”

- Often on index cards
  - Allows easy sorting

- Can also include:
  - Priority
  - Time estimate
  - Identifier (e.g., 234)
XP: Design

- **“Metaphor”** = capture system concept
  - Just an idea to understand how system works

- Emphasis on **simple design**
  - **YAGNI** = “You Aren’t Going to Need It”

- If you need to redesign → **refactoring**

- May use other design artifacts (UML diagrams, etc.)

- **Spike** = very simple program/prototype to explore solutions
  - Also helps team get better estimates
  - **Architectural spike** = helps develop project metaphor
XP: Teams and Process

- **Small** teams with *customer continuously present* on-site

- **Short cycle time** → no more than 3 weeks

- **Project velocity** = how much work you are getting done per iteration
  - Add up all the time estimates for the user stories completed in the iteration

- **Collective ownership** of product
  - Anyone can work on anything
  - **Pair programming** often employed
    - Two programmers sit next to each other and work on same problem/code

- Coding standards established by team

- Emphasis on 40-hour work week
XP: Quality Assurance

• Quality assured through:
  ○ Test-first approach
  ○ Continuous integration

• Test-first approach
  ○ Customers help define **acceptance tests** for each requirement/user story
    ▪ Tests may/should be automated
  ○ Pass test → requirement met

• Continuous integration
  ○ Make simpler but complete versions as you go along
  ○ Build on previous iterations
XP: Development Lifecycle

Extreme Programming Project


Copyright 2000 J. Donvan Wells

http://www.extremeprogramming.org/map/project.html
XP: Comments

- **Advantages:**
  - Works very well with small applications

- **Disadvantages:**
  - Scaling an issue → max. 20-person team
  - Simple design/YAGNI → inappropriate for stable systems with predictable evolution
  - Issues when teams not collocated
Dynamic Systems Development Method (DSDM)
Dynamic Systems Development Method (DSDM)

- Strong emphasis on project management activities
  - Framework for developing software rather than particular method
    - E.g., can be integrated with XP and Scrum
- Developed and run by non-profit *DSDM Consortium*
  - [http://www.dsdm.org/](http://www.dsdm.org/)
  - First formed in 1994
  - Most users in Europe and Great Britain
- Renamed “DSDM Atern” in 2007
DSDM: Phases

- **Pre-project**
  - Describe business problem clearly, plan for Feasibility phase

- **5 phases is DSDM Atern process:**
  - Feasibility
  - Foundations
  - Exploration
  - Engineering
  - Deployment

- **Post-project**
  - Disband team
  - Product in maintenance posture
  - Assess whether you solved original problem
DSDM: Phases

- **Feasibility + Foundations**
  - Done in sequence
  - Determine scope of work
  - Develop prioritized requirements
  - Risk/cost estimation
  - Scheduling
  - Short → no more than a few weeks

- **Exploration**
  - Iterative and incremental
  - Develop partial solution

- **Engineering**
  - Iterative and incremental
  - Make partial solution robust enough for use

- **Deployment**
  - Increment put into use
  - Either:
    - *Scope larger* → go back to Foundations
    - *More requirements to implement* → go back to Exploration
    - *Technical issues* → go back to Engineering
    - *All requirements done* → go to Post-Project
DSDM: Planning and Scheduling

- **Planning in each phases**
  - Evolves based on results from previous phases

- **Timeboxing** used
  - Schedule and cost $\rightarrow$ constant
  - Features to implement $\rightarrow$ variable

- **Requirement priority assigned through MoSCoW:**
  - Must have
  - Should have
  - Could have
  - Want

- **Scripts $\rightarrow$ define management activities**
- **Work products $\rightarrow$ defined/described for each phases**

- **Provides risk management process**
DSDM: Teams and Roles

- Designed for small teams
  - But can be scaled to almost any size

- 10 project roles
  - Clear defined roles and responsibilities
  - May be combined
DSDM: Comments

- Strong infrastructure and traditional feel
  - Easier to implement for businesses used to heavier processes

- Uses risk management \(\rightarrow\) more Agile than traditional disciplined processes
Rational Unified Process (RUP)
RUP

- **Rational Unified Process (RUP)**
  - Developed concurrently with UML (Unified Modeling Language) by Rational Corp. (now division of IBM)
  - Based on risk-driven spiral processes and economics of software development to streamline disciplined software methods
RUP: Tenets

- Four fundamental tenets of RUP:
  - Reduce size and complexity of what needs to be developed
  - Improve development process
  - Create more proficient teams
  - Use integrated tools to exploit automation
RUP: Phases

- Four-phase life cycle → each phase incorporates multiple iterations or spirals
  - Inception
  - Elaboration
  - Construction
  - Transition

- Activities overlap, but more emphasis in certain phases

- “Serial in the large; Iterative in the small”

- Exit phase IF:
  - Meet phase-specific exit criteria
    - Milestones for each phase
  - Have reasonable, detailed plan for next phase
RUP: Phases

Inception
- Define project scope
- Estimate cost and schedule
- Define risks
- Develop business case
- Prepare project environment

Elaboration
- Specify requirements in greater detail
- Identify architecture
- Validate architecture
- Evolve project environment
- Staff project team

Construction
- Model, build, and test system
- Develop supporting documentation

Transition
- System testing
- User testing
- System rework
- System deployment

Lifecycle Objectives (LCO)
- Scope concurrence
- Initial requirements definition
- Plan concurrence
- Risk acceptance
- Process acceptance
- Business case
- Project plan

Lifecycle Architecture (LCA)
- Vision stability
- Requirements stability
- Architecture stability
- Risk acceptance
- Cost and estimate acceptance
- Realistic chance to succeed
- Project plan

Initial Operational Capability (IOC)
- System stability
- Requirements stability
- Prepared stakeholders
- Risk acceptance
- Cost and estimate acceptance
- Project plan

Product Release (PR)
- Business acceptance
- Operations acceptance
- Support acceptance
- Cost and estimate acceptance

Copyright 2004-2005
Scott W. Ambler

Modified from:
http://www.enterpriseunifiedprocess.com/essays/productionPhase.html
RUP: Comments

- Generally viewed as plan-driven, heavy process
  - Large volume of process guidelines
  - “Tailor-down” approach

- Does have Agile attributes in philosophy
  - However, difficult to fit RUP Classic to smaller projects

- Two supported versions:
  - RUP Classic
  - RUP for Small Projects

- Addresses business and economic factors
  - Not usually specifically addressed by other methods
Team Software Process (TSP)
Team Software Process (TSP)

- Tailorable, plan-driven development process for building industrial-strength software with teams
- Used in conjunction with Personal Software Process (PSP)
- Developed by Watts Humphrey (SEI)
TSP: Objectives

Five objectives of TSP:

1) Build self-directed teams that:
   - Plan/track their work
   - Establish goals
   - Own their processes and plans

2) Show managers how to:
   - Coach/motivate teams
   - Help them sustain peak performance

3) Accelerate software process improvement by making SW-CMM/CMMI Level 5 behavior normal and expected

4) Provide improvement guidance to high-maturity organizations

5) Facilitate university teaching of industrial-grade team skills
TSP: Iterative

- Iterative process
  - Begins with formal **Launch event**
    - 4-day meeting to define objectives, roles, estimates, risk managements, processes, etc.
  - Initial phase
  - Two-day **Relaunch events** for each phase until project is complete
    - Define roles, tasks, goals, and quality measures for following phase
TSP: Pros and Cons

**Advantages:**
- Very effective in industrial and academic settings
- Works with projects up to 20 people
  - More people → use “team of teams”
- Very thoroughly defined process

**Disadvantages:**
- Team develops their own version of processes → if team is undisciplined, this can be problematic

**Can be made similar to Agile-based approaches**
Feature-Driven Development (FDD)
Feature-Driven Development (FDD)

- By Jeff DeLuca and Peter Coad
  - From experiences on complex commercial lending application for large Singapore bank in 1997-1998
- Focuses on:
  - Simple process
  - Efficient modeling
  - Short, iterative cycles (build “feature-by-feature”) → deliver value to customer
- Depends heavily on good people for domain knowledge, design, and development
- Goal: process in background to support, rather than drive, the team
FDD: Phases

- Assumes requirements already well captured and understood
- Five phases:
  1) Develop an overall model of product to capture breadth of domain
     - Class architecture with notes about rationale
  2) Establish list of features based on business needs
     - List of items customer finds valuable
     - Feature template: <action> <result> <object>
       - E.g., “Calculate the total of the sale”
  3) Create development plan based on list of features
     - Continuously refined → also defines order of development
  4) Develop design packages and work packages for features assigned to current iteration
     - Bundle features by technical reasons (rather than business reasons)
  5) Build features in software
     - Implement, inspect, test at unit level
     - When complete → integrate into current build of overall system
- Phases (4) and (5) repeated at each iteration of project
- Would be incremental waterfall, EXCEPT that relies on “good people” to recognize when to go back to previous phases
FDD vs. XP

- **Code ownership**
  - FDD → classes assigned to individuals
  - XP → collective ownership of code

- **Architecture**
  - FDD → focus on architecture and “getting it right the first time” (Phase 1)
  - XP → simple design and YAGNI

- **FDD then:**
  - Stronger with stable, predictable systems
  - Weaker with unpredictable, “architecture-breaker” changes
FDD: Comments

• Upfront architecture and planning allows:
  ○ Larger teams
  ○ Parallel development

• Emphasis on “good people” → needs stronger complement of Cockburn Levels 2 and 3 people
  ○ Compensates for sudden architecture changes and lack of emphasis on risk

• Specifically defines process and roles in an “object-oriented” mindset
CMM

CMM = Capability Maturity Model

- Organized set of practices to improve *process capability* in one or more discipline areas
  - *Example areas:* software engineering, systems engineering
SW-CMM

- Capability Maturity Model for Software (SW-CMM)
  - Gained prominence in late 1980s and early 1990s
  - Based on highly-disciplined software processes in aerospace and commercial industries
  - Large community of users
  - Mostly management model
    - Only one part for technical practices
    - Light on people factors (covered in People-CMM)
SW-CMM: History

- U.S. Air Force wanted a way to evaluate companies that developed their software

- 1986 – formal collaboration with:
  - Software Engineering Institute (SEI) of Carnegie-Mellon University
  - MITRE Corporation

- Goal: produce framework for U.S. federal government to assess capabilities of software development contractors
SW-CMM: Creation and Development

- 1987 – *Watts Humphrey* (SEI) and team publish initial framework and a maturity questionnaire
  - Watts Humphrey → “Father of Software Quality”
- 1991 – Version 1.0 of SW-CMM
SW-CMM Framework

- SW-CMM includes:
  - Maturity Levels 2-5
    - Level 1 is the default mess you start with
  - Key Process Areas (KPAs)
    - Describe minimum activities and common features of a process for a particular area
    - I.e., the stuff you should be doing at a given maturity level

- The five levels of maturity:
  - 1) Ad hoc, chaotic
  - 2) Repeatable
  - 3) Defined
  - 4) Managed
  - 5) Optimizing
SW-CMM: Levels 1-2

1) Initial
   - Ad hoc, chaotic
   - If there are any standard practices, chucked during crisis
   - Success depends on heroes and legends
     - ...who probably will leave the company eventually and take their methods with them

2) Repeatable
   - Basic and consistent project management processes
     - Track cost, schedule, and functionality
   - Development process similar only for similar previous projects
     - May vary from project to project
SW-CMM: Levels 3-4

3) Defined
- Management and engineering process is documented and standardized
  - Used throughout organization for all projects
- Minimal learning curve for people moving to new teams/projects

4) Managed
- Have detailed measurements of software process and product quality
- Use measurements to drive strategic analysis
- Process and products are quantitatively understood and controlled
5) Optimizing

- Continuous, pro-active process improvement
  - Quantitative analysis of the process
  - Piloting innovative ideas and technologies
- Situation-based optimization
- (Most companies don’t get this far...)

SW-CMM: Level 5
While SW-CMM was gaining ground, other standards were being developed

- ISO 15504 and 12207 → similar to SW-CMM
- CMM or CMM-like structures for system engineering, security, personnel, and other disciplines

Lots of overlap, so it seemed like there would be a way to combine them...

- Hence why CMMI was developed
CMMI

- Capability Maturity Model Integration (CMMI)
  - 1997 – developed by SEI-CMU, cosponsored by:
    - Department of Defense
    - National Defense Industrial Association
  - 2010 – CMMI Version 1.3
CMMI Goals

- **Goals:**
  - Provide standard for CMMs
    - Includes systems, not just software
  - Incorporate new understanding of development practices
    - Revises and augments SW-CMM → now considered successor to SW-CMM
  - Attempts to harmonize with ISO standards

- **CMMI is NOT:**
  - An engineering development standard
  - A development life cycle

- **It doesn’t tell you specifically HOW to do things, but does tell you:**
  - What sorts of things you should be doing
  - How you can measure, track, and improve on those things
CMMI defines set of requirements for engineering processes

- Two kinds of information
  - Process Areas (PAs)
  - Generic Practices (GPs)
CMMI Process Areas

- CMMI Process Areas (PAs) include requirements for:
  - Basic project management and control
  - Basic engineering life cycle processes
  - Fundamental support processes
  - Process monitoring and improvement processes
  - Integrated development using teams

- Also includes a few that are not in SW-CMM to be more “Agile-friendly”
CMMI Generic Practices

- Generic Practices
  - Support improvement of processes established under the Process Areas
  - Have 6-level capability scale
    - 1) Not performed
    - 2) Performed
    - 3) Managed
    - 4) Defined
    - 5) Quantitatively managed process
    - 6) Optimizing
CMMI Capability Scale Levels 1-3

1) Not performed
   - Not even doing the basics

2) Performed
   - Just doing it

3) Managed
   - Basic project management
4) Defined
- Process standardization

5) Quantitatively managed process
- Uses quantitative measures to monitor and control selected subprocesses

6) Optimizing
- Continuous process improvement
CMMI: How to look at it

- CMMI can be viewed in one of two ways:
  - Staged
    - Process Areas organized by maturity levels, like SW-CMM
  - Continuous
    - Process Areas evaluated and improved separately, similar to ISO
Personal Software Process (PSP)
Personal Software Process (PSP)
- Also developed by Watts Humphrey
- Focuses on software development by *individuals*
  - Improves individual programming skills
PSP: Levels

- PSP has four levels of personal process: Levels 0 – 3
  - PSP0
    - Establish personal baseline using basic measurements and reports
    - PSP0.1 → adds coding standard, size measurement, development of a personal process improvement proposal
  - PSP1
    - Adds planning and disciplined approach to testing
    - PSP1.1 → adds task and schedule planning
  - PSP2.0
    - Centered around quality
    - Introduces code and design reviews
    - PSP2.1 → adds design templates
  - PSP3
    - Sets programmer on cyclical development and improvement cycle
PSP: Summary

- Makes bad habits perceptible and subject to conscious mental control
- Shown effective in academic and industrial venues
- Some don’t find continuing the process as effective
These approaches cover two different areas:

- TSP $\rightarrow$ team processes
- PSP $\rightarrow$ individual processes
Cleanroom
Origins of Cleanroom

- Developed by Harlan Mills at IBM

- Focus on **defect-free** code
  - Program = complex mathematical function
  - Ergo, can mathematically verify it works \(\rightarrow\) “certified” that it is reliable
Cleanroom Definition

Complex software development discipline including processes for:
- Planning
- Specification
- Design
- Verification
- Coding
- Testing
- Certifying software
Cleanroom Processes

- System behavior initially “black box”
  - Stimuli, responses, and transition rules
  - Later becomes “clear box”

- Stringent reviews of design and code BEFORE compilation

- Defect measures maintained from that point forward

- Incremental approach
  - Strict construction rules

- Uses statistical testing → system testing based on probabilistic estimated of functional usage
  - Need tools that support this
  - Useful idea outside of Cleanroom approach
Cleanroom Pros and Cons

- **Advantages:**
  - Excellent for developing highly reliable software

- **Disadvantages:**
  - Need highly skilled practitioners
  - Difficult to scale up
  - Problematic in highly dynamic environments
References

- CMMI FAQ: [http://www.cmmifaq.info](http://www.cmmifaq.info)
- User stories: [http://www.agilemodeling.com/artifacts/userStory.htm](http://www.agilemodeling.com/artifacts/userStory.htm)
- Crystal: [http://alistair.cockburn.us/Crystal+methodologies](http://alistair.cockburn.us/Crystal+methodologies)
- Crystal Methods: [http://alistair.cockburn.us/Crystalmethods180.ppt](http://alistair.cockburn.us/Crystalmethods180.ppt)
- DSDM Atern Lifecycle: [http://www.dsdm.org/content/6-lifecycle](http://www.dsdm.org/content/6-lifecycle)
- Manager’s Intro to RUP: [http://www.ambysoft.com/downloads/managersIntroToRUP.pdf](http://www.ambysoft.com/downloads/managersIntroToRUP.pdf)