FACULDADE ENGENHARIA
Departamento de Informática

Software Process#2

(adapted from lecture notes of the CSCI 3060U - Software Quality Assurance unit, J.S. Bradbury, J.R. Cordy, 2018)

Agile Methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.

- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

The Principles of Agile Methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Agile Development

- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- Minimal documentation – focus on working code

Extreme Programming

Is XP a philosophy/way of life or a software development process?

Probably a bit of both...

Extreme Programming

A Philosophy

- XP has values, principles and practices
- Values of XP: Communication, Simplicity, Feedback, Courage, Respect…

"Values bring purpose to practice"

Extreme Programming

A Philosophy

- Principles of XP:
  - Humanity - software is developed by people
  - Economics - software costs money
  - Mutual Benefit - software activities should benefit everybody
  - Self-Similarity - try reusing solutions across projects
  - Improvement - perfect software doesn't exist
  - Diversity - different skills benefit a software team
  - Quality - can not be sacrificed
  - Other Principles: Flow, Opportunity, Redundancy, Failure, Baby Steps, Accepted Responsibility, Reflection

Extreme Programming

A Modern, Lightweight Software Process

- Extreme Programming, or XP, is a new lightweight process
- Originally used for small- to medium-sized software projects (Although it can scale to larger projects)
- Designed to adapt well to the observed realities of modern software production:
  - short timelines
  - high expectations
  - severe competition
  - unclear and rapidly changing requirements
- Based on the idea of continuous evolution
- Very practical – based largely on simplicity, testing
- In spite of its brash, undisciplined, "fun" presentation, solidly based on the software disciplines and processes of the past

What's So Extreme About It?

Why is it called Extreme?

- When first conceived, the idea was to take the best practices of good software development to the limit (the extreme)
  - if code reviews are good, review code all the time
  - if testing is good, test all the time
  - if design is important, design all the time
  - if simplicity is good, always use the simplest solution possible
  - if architecture is important, refine architecture all the time
  - if integration is important, integrate all the time
  - if short iterations are good, use shortest iterations possible

Oh No! Not Yet Another Process ...

Why Make a Different Approach?

- XP was born of the dissatisfaction of programmers with the actual situation in most software development environments
- Frustration the lack of time to test adequately because of the rush to get new software and new versions out quickly
- Dissatisfaction with the lack of ongoing advice and social support for difficult technical decisions, and management blame for decisions that do not turn out well
- Worry about lack of connection between planning and design activities and actual source code
- Worry about the communication gap between management and technical staff

Properties of Extreme Programming

Characteristics of XP

- continuing feedback from short cycles
- incremental planning that evolves with the project
- responsive flexibility in scheduling
- heavy and continuous use of testing and test automation
- emphasis on close and continuous collaboration and communication
- use of tests and source code as primary communication media (communication at programmer's level)
- evolutionary model from conception to retirement of system
- emphasis on small, short term practices that help yield high quality long term results

Attacking Risks Before They Arise

Addressing Risk

- XP tries to explicitly address the greatest risks to software development projects actually observed in practice

(1) Schedule Slips

- Software isn't ready on the scheduled delivery date
- Addressed in XP by short release cycles, frequent delivery of intermediate versions to customers, customer involvement and feedback in development of software

(2) Project Cancellation

- After several schedule slips, the project is cancelled
- Addressed in XP by making the smallest initial release that can work, and putting it into production early – thus establishing credibility and results

Attacking Risks Before They Arise

(3) System Defect Too High, or Degrades with Maintenance

- Software put in production, but defect rate is too high, or after a year or two of changes rises so quickly, that system must be discarded or replaced
- Addressed in XP by creating and maintaining a comprehensive set of tests run and re-run after every change, so defect rate cannot rise
- Programmers maintain tests function-by-function, users maintain tests system feature-by-system feature

(4) Business Misunderstood

- Software put in production, but doesn't solve the problem it was supposed to
- Addressed in XP by making customer an integral part of the team, so team is continually refining specification to meet expectations

Attacking Risks Before They Arise

(5) Business Changes

- Software put in production, but business problem it is designed for changes or is superseded by new, more pressing business problems
- Addressed in XP using short release cycles and by having customer as an integral part of the team
- Customer helps team continually refine specification as business issues change, adapting to new problems as they arise -programmers don't even notice

(6) Featuritis (or False Feature Risk)

- Software has a lots of potentially interesting features, which were fun to implement, but don't help customer make more money
- Addressed in XP by addressing only the highest priority tasks, maintaining focus on real problems to solve

Attacking Risks Before They Arise

(7) Staff Turnover

- After a while, the best programmers begin to hate the same old program, get bored and leave
- In XP programmers make their own estimates and schedules, get to plan their own time and effort, get to test thoroughly
- Less likely to get frustrated with impossible schedules and expectations
- In XP emphasis is on day to day social human interaction, pair and team effort and decisions
- Less likely to feel isolated and unsupported

XP in Practice

We will now look at the actual practices of the XP process.

- In XP, primary practices are good practices to start with when beginning with XP (we will focus mainly on these in our project)
- In XP, corollary practices are for experience XP teams. These practices are dangerous without first mastering the primary practices.

XP in Practice - Planning Practices

Stories

- Story – “a unit of customer-visible functionality”
- Each story should have
  - a name,
  - a short description (written or graphical), and
  - an estimate of the implementation effort required.
- Usually written on index cards and placed on a wall in the office.

XP in Practice - Planning Practices

Cycles and Slack

- Weekly Cycle – Plan one week at a time.
  - Have a weekly meeting to
    - discuss last week's actual vs. expected progress
    - pick stories to implement this week. Each story is broken into tasks (effort for each task is estimated).
- Quarterly Cycle – Plan one quarter at a time
  - Have a quarterly planning meeting to
    - reflect on the team and project with respect to large goals.
    - plan themes for the quarter and pick stories for each theme.
- Slack - plan for slack
  - Build slack into plan. Don't under estimate the effort to implement stories.

XP in Practice - Planning Practices

The Planning Game - Business vs. Technical Constraints

- The Planning Game refers to the practice of having a continuous dialog between business and technical people on the project
- In weekly meetings, business people bring business constraints, and technical people bring technical constraints
- Business people bring issues of scope, priority, releases
- Technical people bring estimates, consequences, scheduling
- Forces the project members to continually balance between what is possible (the technical aspects) and what is desirable (the business aspects)

XP in Practice - Planning Practices

Plan for Small Releases

- Small Releases refers to the practice of addressing only the most pressing business requirements, and getting them addressed by releasing a new version quickly
- Means that we should bring the first version into production as quickly as possible
- Means that we should shrink the cycle to the next version as much as possible

XP in Practice – Programming Practices

Pair Programming

- Pair Programming refers to the practice of having all production code written with two people working together on one terminal
- One partner works tactically, on the specific part of the code (e.g. method) being coded at the moment
- The other partner works strategically, considering higher level issues such as:
  - is this approach going to work?
  - can we simplify this by restructuring?
  - what other tests do we need to address here?

XP in Practice – Programming Practices

Test First Programming

- The only required program features are those for which there is an automated test
- Always create tests first, and treat them as the goal (specification)
- Programmers create unit tests (tests for each method or segment of code)
- Customers create functional (acceptance) tests (tests that check that the product has the required functionality)

XP in Practice – Programming Practices

Incremental Design

- Improving and work on the design of the system every day
- Refactoring is part of incremental design and refers to the practice of continually looking for ways to simplify the architecture and coding of the system as new features and changes are made
- When a new feature or change is needed, we first look to see if there is a way to rearchitect the system to make it easier or simpler to add – if so, we rearchitect first
- Once the new feature has been added or changed, we look to see if the resulting new program can be simplified by rearchitecting or merging similar code

XP in Practice – Programming Practices

Coding Standards

- Coding Standards are project-wide conventions about the coding of programs
- Necessary since everyone is responsible for all of the code, and may have to read or change any part of it at any time
- Usually specifies:
  - commenting standards, e.g., every method must have a comment of the form …
  - naming conventions, e.g., variables representing dates will always be named ending in “Date”, all constants will be named with a two letter prefix indicating their business type, etc.

XP in Practice - Integration Practices

Continuous Integration

- In XP, new code is always integrated and tested within a day
- Changes are not allowed to go on without being continually tested in context, to catch integration failures before they happen

Ten-Minute Build

- Automatically building the entire system and running all the tests should take no more than 10 minutes
- A short build means more chances for feedback.

XP in Practice - General Practices

The following general practices are primarily related to the environment of an XP team.

Sit Together

- The whole team should work in an open space.

Whole Team

- The whole team means having people with the necessary skills and the right attitude.

Informative Workspace

- Make the workspace about work (e.g. visual display of project information such as stories).

XP in Practice – Corollary Practices

Corollary Practices by Category

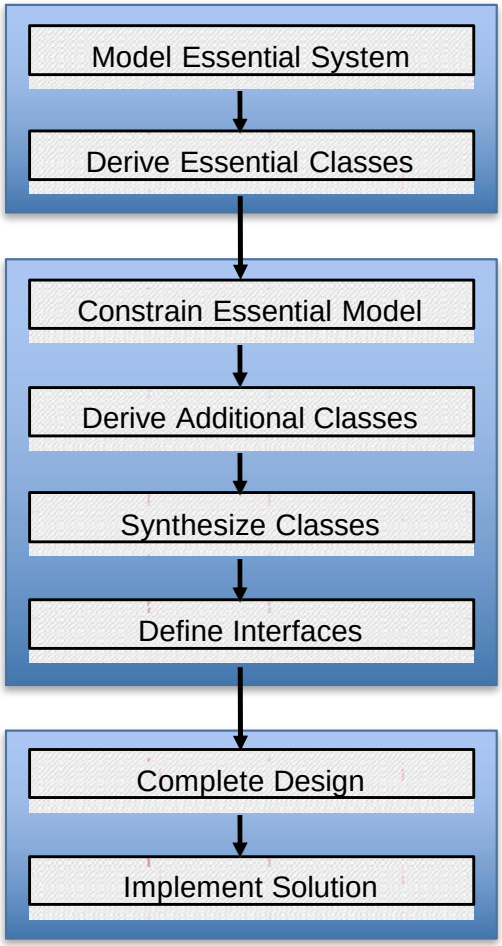
- Business practices: negotiated scope contract, pay-per- use, daily deployment.
- Programming practices: single code base, shared code, code & test.
- Team practices: team continuity, shrinking teams, real customer involvement, root cause analysis.

The Object-Oriented Development Process

OO Process Models

- Many OO process models are proposed, based on OOAD (*"Object-Oriented Analysis and Design"*)
- All have three major phases:
  - Analysis – model the "essential system" to represent user requirements, and design implementation-independente "essential classes"
  - Design – constrain and refine essential classes to be implemented on particular implementation environment, derive additional classes
  - Implementation – define class interfaces and implementation methods, then code and unit test all classes

The OO Development Process



Analysis

Design

Implementation

The OO Development Process – Analysis

(1) Model the Essential System

- Create a "user view" of the system
- Model essential activities and essential solution data, how related
- Quality control - requirements reviews (inspection)

(2) Derive Essential Classes

- "Carve" out candidate essential classes from the essential model using data-flow diagrams, process and data specifications
- Quality control - design reviews (inspection)

The OO Development Process – Design

(3) Constrain the Essential Model

- Modify essential model to fit within constraints of target implementation environment
- Map essential activities and data to implementation processors (hardware/software) and containers (memory/files)

(4) Derive Additional Classes

- Additional classes and methods specific to implementation environment added to support additional activities added while constraining the essential model

The OO Development Process – Design

(5) Synthesize Classes

- Essential classes and additional classes refined and organized into a class hierarchy – final classes chosen to maximize reuse
- Quality control: design review (inspection)

(6) Define Interfaces

- Class definitions written for final classes

The OO Development Process - Implementation

(7) Complete Design

- Design of "implementation module" completed
- Implementation module specifies methods such that each provides a single cohesive function
- Quality control: design review

(8) Implement Solution

- Implementation of classes and methods is coded and validated
- Quality control: unit testing (class-wise)

Drawbacks of the OO Development Process

Not Yet Mature

- There are several competing OO development processes which disagree on steps
- Not yet much experience on large projects, seems to work well on small things
- Little practical detail in step descriptions, difficult for new users to understand

Drawbacks of the OO Development Process

Role of Testing

- Development process missing intermediate versions, almost all testing delayed to final implementation stage

Architectural Inflexibility

- Process assumes that overall architecture can be designed in the requirements phase, allows little architectural flexibility in design and implementation steps

Software Process Evaluation

How Can We Evaluate Software Processes?

- There are several methods and standards for software process evaluation

- Most are aimed at improving existing development processes as they are applied – called maturing them

- Idea is that, as a company or team gains experience with a process, they continually improve it to make it better in their use

The Defect Prevention Process

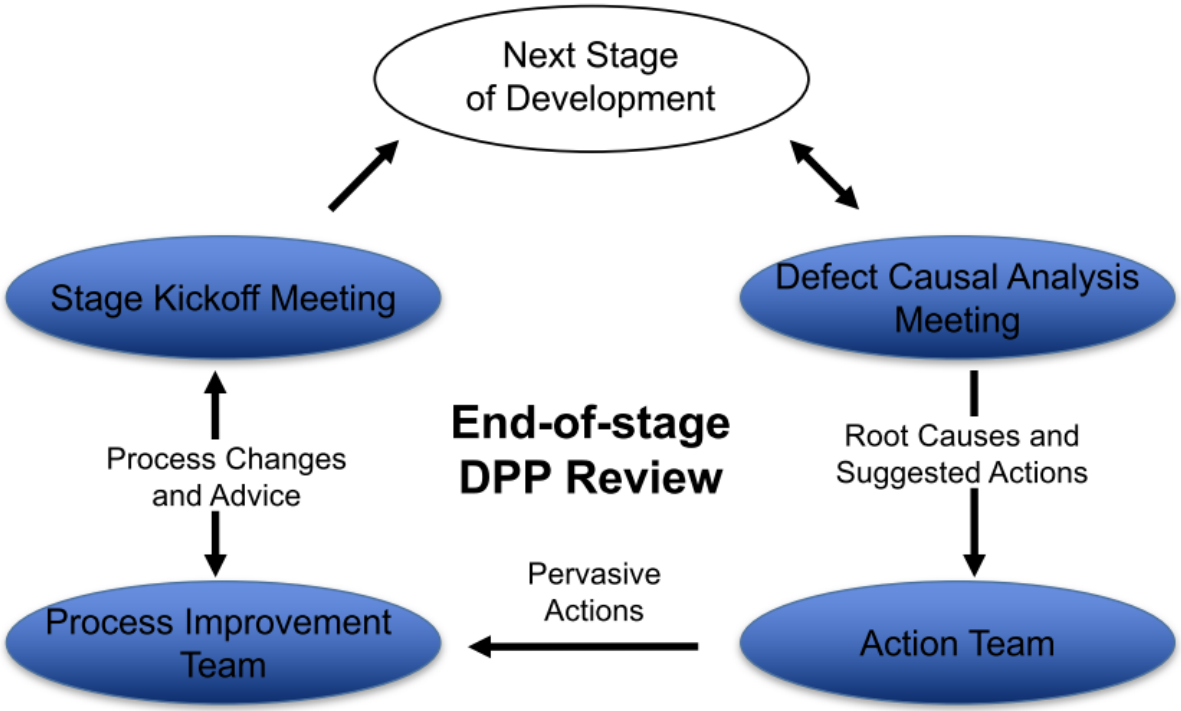
DPP - Defect Prevention Process

- DPP is not itself a software development process, but rather a process for continually improving the development process
  - Modelled on quality assurance techniques used in Japan for decades
- Based on three simple steps:
  - analyze existing defects or errors to trace their root causes in the process (i.e., how they were missed)
  - suggest preventive actions to eliminate the defect root causes from the process
  - implement the preventive actions to improve the process

The Defect Prevention Process

Formal DPP Reviews

- First used at IBM Communications Programming Lab (1985)



The Defect Prevention Process

(1) Defect Causal Analysis Meeting

- At end of each stage of development, review and analyze defects that occurred in that stage in a short meeting
- Developers trace root causes of errors, suggest possible actions for preventing similar errors in future

(2) Action Team

- Action team has cross-organization members
- Evaluates suggested actions, initiates actions across the organization, including development team actions

(3) Process Improvement Team

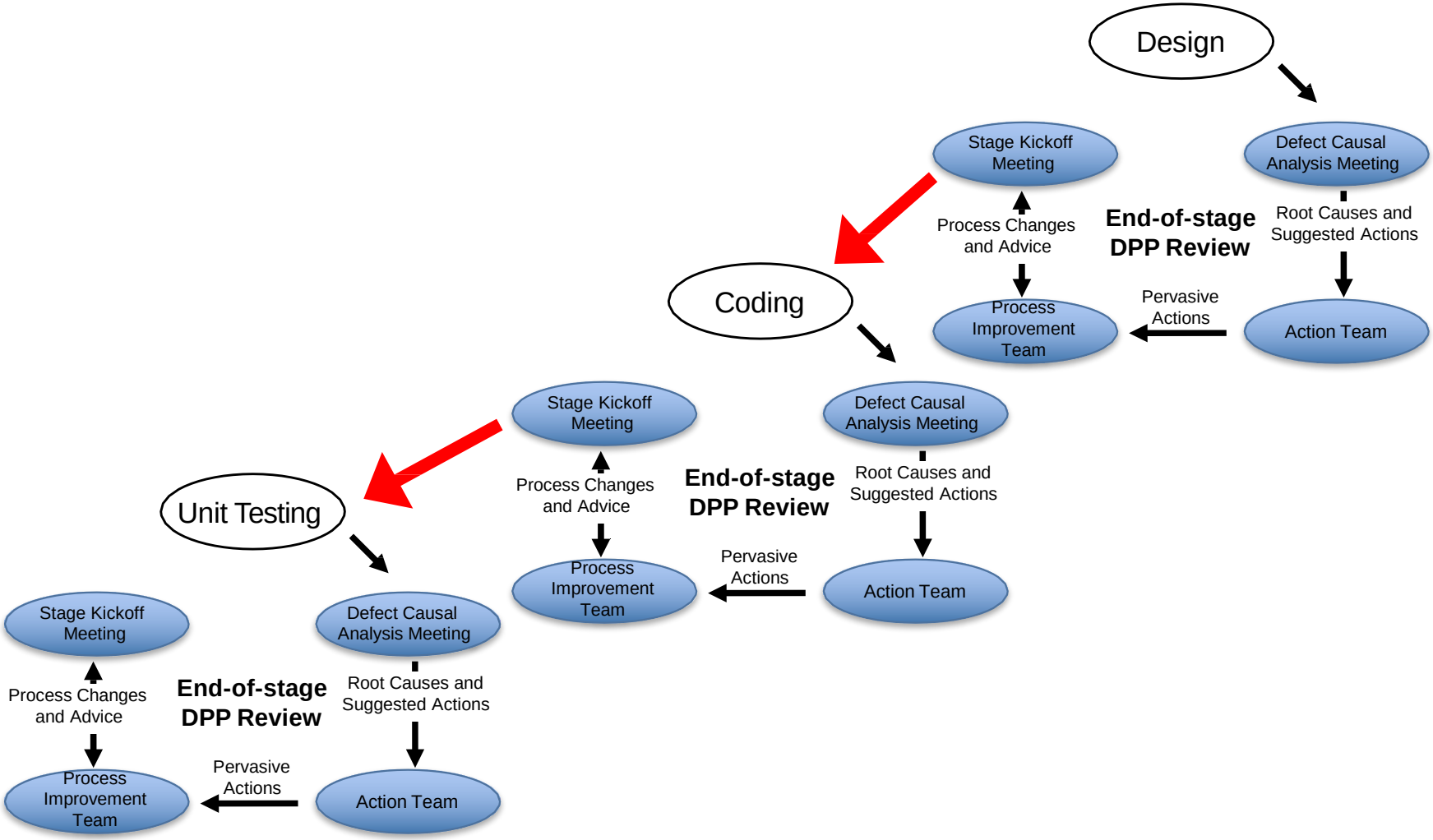
- Members of the development team
- Implements process changes and provides advice for next stage of development

(4) Stage Kickoff Meeting

- Development teams meet to review process changes and re-emphasize focus on quality

FACULDADE ENGENHARIA
Departamento de Informática

DPP Applied to the Waterfall Model



Process Quality Standards

Software Process Assessments and Standards

- There are two kinds of quality standards:
  - Maturity Models
    - Maturity models attempt to measure how well developed (mature) the software process in a particular organization is, and thus how likely it is to produce quality results
  - Certification Standards
    - Certification standards measure an organization's software process against a defined standard, and certify the organization if its process meets the standard

Capability Maturity Model (CMM)

- Capability Levels – *"characterize the state of the organization's processes relative to an individual process area."*
- Maturity Levels – *"characterize the overall state of the organization's processes…"*
- Levels are evaluated using SCAMPI – Standard CMMI Appraisal Method for Process Improvement

[Source: CMMI for Development v1.3, CMMI Institute]

Capability Maturity Model Integration (CMMI)

(0) Capability Level 0 - "Incomplete"

- Process is not used or only partial completed

(1) Capability Level 1 – "Performed"

- Process goals are satisfied

(2) Capability Level 2 – "Managed"

- Process is monitored and evaluated

(3) Capability Level 3 – "Defined"

- Process is a managed and geared specifically towards organization

Capability Maturity Model Integration (CMMI)

(1) Maturity Level 1 - "Initial"

- ad-hoc and chaotic

(2) Maturity Level 2 - "Managed"

- planned process, projects are evaluated against process description

(3) Maturity Level 3 - "Defined"

- set of standard organization processes that are well understood

(4) Maturity Level 4 - "Quantitatively Managed"

- quantitative, quality and process performance are both measured

(5) Maturity Level 5 - "Optimizing"

- quantitative assessment drives continual process improvement

SPR Maturity Assessment

Software Productivity Research (SPR) Assessment

- Much like CMM, but focuses more broadly on corporate strategy and tactical issues as well as CMM's issues of software organization and process
- Also uses a questionnaire, but has 400 questions as opposed to CMM's 85, and uses a five-point scale instead of yes-no answers
- Assessment uses measures such as:
  - quality and productivity measurements
  - experience of programmers in defect removal and testing-project quality and reliability targets
  - defect removal history in each phase (design, coding, testing, release)

ISO 9000 Family of Standards

ISO = International Organization for Standardization

ISO 9000 Family of Standards

- Originally developed in 1987 and subsequently revised ~5-10 years
- A set of standards and guidelines for quality assurance management
- Many customers, especially in Europe, require ISO 9000 registration of their suppliers
- Companies become ISO 9000 "registered" as a result of a formal audit by ISO

ISO 9000 Family of Standards

ISO 9000 Family of Standards

- ISO 9000 standards are documentation-based:
  - every aspect of every step of every process must be backed up by formal documents in a precisely defined format keeping records of how processes are applied
- Standards are complex, detailed and stringent
- Example: The documentation standard goes so far as to specify:
  - owner of document must be specified on title page
  - distribution of document must be controlled with an archived master copy, distribution record book, etc.
  - version level must be clearly identified
  - all pages must be consecutively numbered
  - total number of pages must be indicated on title page
  - procedure for destruction of obsolete documents must be documented

ISO 9000 Family of Standards

ISO 9000 Family of Standards

- Historically **60-70%** companies fail the ISO audit the first time
- Most software companies are deficiente in corrective actions and document control
- Companies take steps to meet the standards in these áreas and usually can be registered on the second try

ISO 9000 Family of Standards

ISO 9000 Family of Standards

▪There are a number of different standards in the ISO 9000 Family including:

- ISO 9000:2015 covers the fundamentals and defines a vocabulary for quality management systems.
- ISO 9001:2015 defines requirements for quality management systems.
- ISO 9004:2009 focuses on improving the effectiveness and efficiency of quality management systems.
- ISO 19011:2011 sets audit guidelines for quality management systems.
- ...

ISO 9000: How does it relate to software quality?

ISO 9000 and Software

- ISO provides documents outlining the application of ISSO 9001:2008 (requirements standard) to software in

ISO/IEC 90003:2014

- Recall, that ISO standards are not static – standards evolve and improve over time
  - Documents applying ISO 9001 to software have also evolved over time: ISO 9000-3:1994, ISO 9000-3:1997, ISO IEC 90003: 2004, ISO/IEC 90003:2014
  - Give the standards for software development
  - Includes topics ranging from customer involvement to testing methods

ISO 25000

- The series of standards ISO/IEC 25000, also known as SQuaRE (System and Software Quality Requirements and Evaluation), has the goal of creating a framework for the evaluation of software product quality
- The series of standards ISO/IEC 25000 consists of five divisions:

[Source: The ISO/IEC 25000 series of standards]

ISO/IEC 25010

- The product quality model defined in ISO/IEC 25010 comprises the eight quality characteristics shown in the following figure:

[Source: The ISO/IEC 25000 series of standards]

Summary

- We have discussed the following software processes:
  - Waterfall Process - the oldest and most common process
  - Prototyping - some recent and popular prototyping-based processes
  - Spiral Model - organizes and generalizes waterfall model
  - Iterative Development Process - based on product subsets
  - Object Oriented Development - a recent and popular model
  - Extreme Programming

Summary

- Software processes can be continually improved using meta- processes such as the Defect Prevention Process
- Software processes can be evaluated with respect to their maturity or by comparison with a process standard
  - Maturity models include CMMI and SPR
  - Process quality standards e.g. ISO 9000, ISO 25000


HOW
WHY
WHERE
WHEN
WHERE
WHY
WHERE
WHO
WHEN
WHAT
WHAT
WHY
WHAT
WHEN
WHO
HOW
WHEN
WHO
WHERE
WHY
HOW