

# Project Management Course Book

T++ Technical Skills Training Program

CUNY Institute for Software Design & Development (CISDD)

New York Software Industry Association (NYSIA)

Aron Trauring

Class 8 October 28th, 2004

# Class Agenda

**1:30-2:45** Unified Process Part I (Presentation)

**2:45-3:00** Break

**3:00-4:00** Project+ Game

**4:00-4:15** Break

**4:15-5:30** Unified Process Part II (Presentation)

# Unified Process

## Historical Background

- Three amigos — three gurus, Grady Booch, Ivar Jacobson, and Jim Rumbaugh
- Rational Systems — now owned by IBM
- RUP or UDP
- High ceremony tradition but influenced by agilists

## Key Practices

- Develop in short time-boxed iterations
- Develop high-risk, high-value elements first
- Focus on customer value — Milestone goals
- Accommodate change early

## Classification

- Formal — high ceremony, although much is optional
- Explicit — lot's of documentation — *ibid.*
- Autocratic — relatively rigid process definitions, especially RUP
- From C1 to L1000

## Four Phases of a Project

1. Inception — establish business rationale and decide on scope
2. Elaboration — detailed requirements, baseline architecture, project plan
3. Construction — IID
4. Transition — move towards deployment

## Inception — Length

- Five minute conversation
- Agile Version — Few days-weeks
- Can be up to several months

**Inception — Purpose**

- High level objectives
- Business case and vision
- Project scope
- Agile version — 10% of requirements captured in detail
- Estimate Elaboration

**Inception — Milestone Goals**

- Agreement on scope, vision and priorities
- Some risks identified
- A plan for elaboration exists

**Inception — Activities and Documents**

- Short requirements workshop
- Top Ten high level requirements list
- First draft of Vision and Business Case documents
- Significant requirements defined in use cases

**Elaboration — Length**

- About 20% of total project
- Several months in long project

**Elaboration — Purpose**

- What are you going to build
- How you are going to build it
- Risk Analysis
- Agile Version — by end, the core of the system and most requirements stabilized

**Elaboration — Milestone Goals**

- The vision, requirements and architecture are stabilized
- The core executable architecture is implemented(A)/prototyped(NA)
- The majority of requirements are defined
- Estimates and coarse grade plans are defined

**Elaboration — Activities and Documents**

- Agile Version — short requirements workshops — one per iteration
- Agile Version — iterations on core, architecturally significant elements
- Non-Agile Version — Prototype of core architecture
- By end, a semi-reliable development plan with reasonable estimates

**Four Risk Types**

1. Requirements Risk
2. Technology Risk
3. Skills Risk
4. Political Risk

**Requirements Risk**

- Use Case Driven
- Get as many use cases as possible — 80% by end
- Domain model — description of world within which system operates
- Documents — conceptual class diagrams and activity diagrams for work-flow
- Model is skeleton — not too much detail but not partial either
- Very small team — conceptual integrity
- Prototype complex, dynamic behaviors to analyze risk
- Use language conducive to prototyping — Python
- Key to success is access to domain experts

**Technology Risk**

- Prototyping — test technologies
- What will happen if a piece of the technology doesn't work?
- What if we can't connect two or more components?
- What is the likelihood of something going wrong and how will we cope?
- How components communicate — class and interaction diagrams
- High level architecture – package diagrams
- How pieces are distributed — deployment diagrams

**Construction — Length**

- Longest part of project — 50%
- Sets of relatively short iterations

**Construction — Purpose**

- System completed and ready for deployment
- Efficient and stable development

**Construction — Milestone Goals**

- System is believed ready to be deployed
- Stake-holders are ready for deployment

**Construction — Activities and Documents**

- Time-boxed iterations
- Stake-holder evaluation
- Detailed documentation generated from code
- Add additional documentation to highlight important stuff
- Package diagrams — logical road map of system
- Class diagrams — graphical table of contents
- Statecharts — complex life-cycle behaviors
- Interaction diagrams — complex class interactions

**Refactoring**

- Addresses software entropy
- Change internal structure of program before adding new functionality
- Have good unit tests
- Make changes in small steps, and test after each change

**Patterns**

- Common ways of doing things
- GOF — Gang of Four - Erich Gamma, Richard Helm, John Vlissides, Ralph Johnson
- Proxy — inter-process communication without knowing where other process is
- Pattern is not just a model — it's a solution to a problem
- Clarifies the problem

- Explains why it solves the problem
- Explains when it works and when it doesn't
- Design patterns most common
- Analysis patterns less formalized

**Transition — Length**

- Varies with project complexity
- Shorter than elaboration but longer than inception
- Time between beta and final release

**Transition — Purpose**

- Verify system ready for deployment
- Deploy

**Transition — Milestone Goals**

- System is deployed
- Users are satisfied

**Transition — Activities**

- Beta testing and Release Candidates
- Parallel operation
- Data conversion
- Training
- Roll-out

**Other UP Terminology**

- Discipline — major areas of concern and activity
- Discipline — Requirements, design, implementation, project management
- Work-products — artifacts — documents 50 or more
- Work-products — Vision, Use Case Model, Risk List, Test Plan
- Roles — Stake-holder, Implementor, Tester, Project Manager
- Practices — Requirements workshop, Coding standards, Model visually

**Core UP Practices**

- Follow the UP guidelines and best practices
- Create at least some UP artifacts
- Conform to UP nomenclature
- Organize iterations by phases

**Core UP Guidelines**

- Attack risks early and continuously or they will attack you
- Deliver value to your customer — early and often
- Stay focused on delivering executable code not specifications or other documentation
- Accommodate change through early development, multiple requirements workshops, etc.
- Prefer component oriented architectures and reuse of components
- Work together as one team
- Quality is a way of life

**Core UP Best Practices**

- Time-boxed iterations
- Cohesive architecture through co-located small team
- Early team members become subproject leaders
- Program high-risk, high-value components first
- Continuously verify quality — test early and often
- Integration testing all the time
- Early usability testing
- Visual modeling
- Find requirements through one day workshops
- Organize requirements with risk, priority and status — use a collector tool
- Track requirements through tool
- Manage change through disciplined configuration management and version control
- Create a change request protocol
- Develop base-lined releases

**Core UP Work-products**

- All are optional, but UP has identified and named all important ones
- Common vocabulary especially important in large organizations — improves communication
- Information abstractions — don't have to be items in a CASE tool — posters, sketches, text

**Core UP Values**

- Project-oriented rather than people-oriented
- Must follow UP guidelines and best practices
- Risk- and value-driven
- Must have clear Vision that articulates stake-holders real needs
- Prefers well-defined process with guidelines on activities, artifacts and individual tasks
- Defined process can be tailored to project size and complexity or Cockburn scale

**Throughput Accounting**

- Inventory (V) is the Use Cases
- Investment is inception phase and portion or elaboration that is still requirements gathering
- Lead Time (LT) — time it takes for Use Case to get through to delivery at end of Transition
- Throughput — dollar value added of software delivered at end of Transition
- Production Quantity (Q) — number of Use Cases delivered out of the Transition
- Production Rate — Q/time period
- Possible to minimize V and LT if use more agile version of UP
- Emphasis on strong ceremony increases OE

**Criticisms of UP**

- Scope-Budget-Schedule model
- Artifacts are to help with estimates
- Not human centric enough
- Doesn't focus on developer as a capacity constrained resource
- Use cases not fine grained so harder to track value flow and increase T and lower LT

**In Defense of UP**

- Defined processes provide QA checklists for novices
- More experienced workers can choose work-products and best practices
- UP makes explicit some useful steps e.g. writing release notes when deploying
- Value of common vocabulary

**Wrong way to use UP**

- Equating phases with SDLC
- Long iterations
- Letting iterations expand — violating the time-box to match scope
- Ending iteration without an integrated and tested baseline
- Elaboration phase creates a throw-away prototype not a product increment
- Forgetting 'Barely Sufficient' when deciding on artifacts
- Predictive planning from early Use Cases and models
- Spending most of the time on the CASE tools