Lecture 1 Why “Software” Project Management

Project management is the planning, scheduling and controlling of project activities to achieve project objectives. The PM goal is to finish the project on budget, on time, feature-complete and with high quality.

Code writing counts for about 40% of software development. It is 44% of the budget cost.

Only 10% of projects are delivered within initial budget and schedule.

Capability Maturity Model (CMM), ranks the Software Development Process of a firm by using 5 levels of maturity: (1) Initial, (2) Repeatable, (3) Defined, (4) Managed, (5) Optimizing.

1. Initial: The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort.
2. Repeatable: Basic project management processes are stablished to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications. It has project management processes. You take a few metrics at the end of the project. But during the project, it’s a black box.
3. Defined: The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization’s standard software process for developing and maintaining software. It has software development processes, like SCRUM. You capture metrics during an iteration and project. But you can’t adjust yourself during a project or iteration.
4. Managed: Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled. Do a lot of metrics. You capture metrics during an iteration and project. And you can adjust yourself during a project or iteration.
5. Optimizing: Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

CMM is now deprecated, and replaced by CMMI which we will see more towards the end of the course.

Lecture 2 Life Cycles, Development Processes and Project Phases

Development process: A model that tells us how to build a software. It defines the stages, its relationships and its activities.

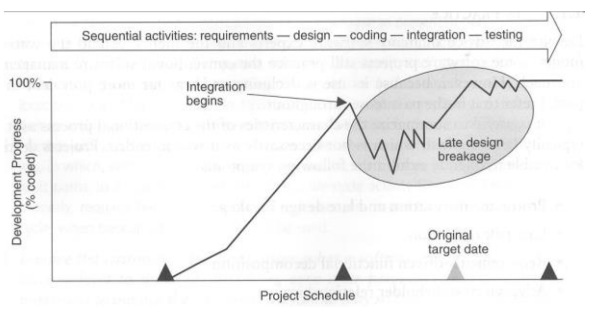
Life cycle (LC) consists of the entire life of a software as a sequence of activities.

The Waterfall process: Requirements, design, implementation, testing, installation, phase-out.

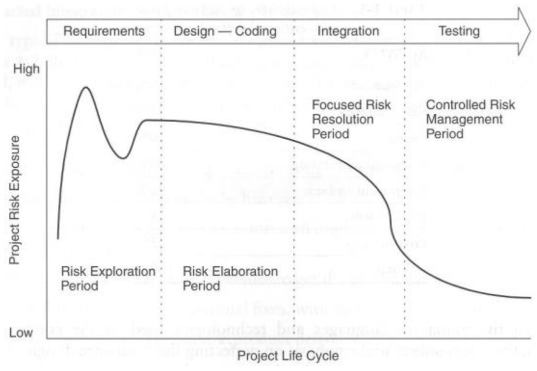
Waterfall works best for things that have been done many times before, as building cars and bridges.

Waterfall problems:

- Late design breakage: Early success via paper design and briefings, but integration nightmares due to unforeseen issues.



- Late risk resolution: Risks become apparent later, and are expensive to mitigate. In a normal project, the bump would not exist, it would always go down. The bump is what was not predicted.



- Requirements-driven functional decomposition: Attempt to provide a precise requirements definition and then implement those exactly. Instead, what should be done, is to get 20 to 50 requirements at the beginning.
- Adversarial stakeholder relationships: Difficulties in requirement specification leads to prolonged paper document exchanges at the beginning.
    - Client: “That was not what I said”.
    - PM: “Yes, that’s what you said, look at the document”.
    - Client: “Yes, that’s what I said, but that’s not what I meant”.
- Focus on documents and review meetings: Only a small percentage of the audience understands the software. Design is considered “innocent until proven guilty”, but the design is always guilty.

Incremental LC: Builds the project in increments. But it’s still similar to waterfall.

Advantages:

- Very fixed chunks.
- Conceptualizable: Because it is smaller, is easier to conceptualize.
- Fast deliverables: Shows progress.
- Cheap outlays: As there are some initial deliverables, it may start working sooner as in a beta version.

Disadvantages:

- Hard to specify builds (not that real compared to Waterfall).
- Integration complex: Has to integrate various builds (not that real compared to Waterfall).

- Degenerations possible: The design was not really good, so it need to be redone (that's acceptable).
- Retrofitting: adding to the existing structure complex.

Rapid Prototyping LC: Develops a prototype at the beginning, the prototype gives a more realistic input to design. But it's still similar to waterfall, but you have the customer feedback very early.

Advantages:

- Quick "product" overview.
- Feedback: try to get it right first time, but recognizes that corrections may be needed.
- Allows the client to "see something".
- Insight into design.

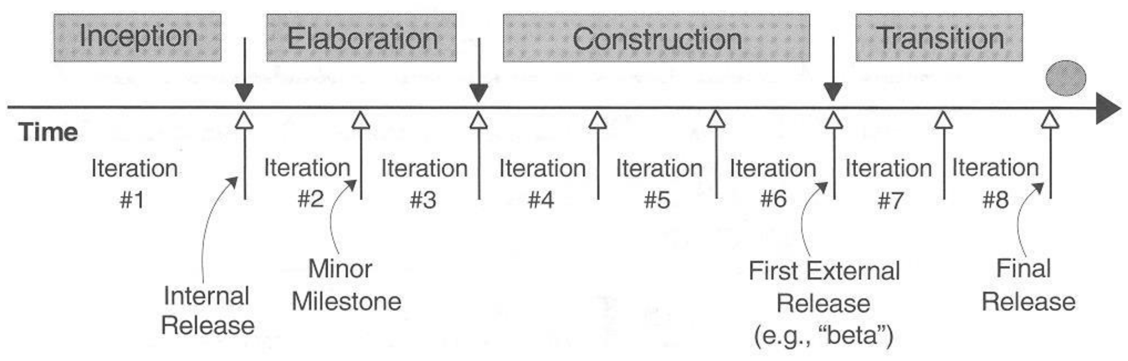
Disadvantages:

- Lengthens time of requirements phase.
- Need to throw away.
- Temptation to release "product" (prototype) early.

Iterative and Incremental (I&I), each iteration: requirements, analysis, design, implementation, testing, show to the customer, then repeat, requirements, analysis, design, implementation, testing, show to the customer, then repeat until done. Principles:

- Architecture-first approach: The central design element. Design and integration first, then production and test.
- Iterative life-cycle process: The risk management element. Risk control through ever-increasing function, performance, quality.
- Component-based development: The technology element. Object-oriented methods, rigorous notations, visual modeling.
- Change management environment: The control element (like git). Metrics, trends, process instrumentation.
- Round-trip engineering: The automation element. Complementary tools, integrated environments.

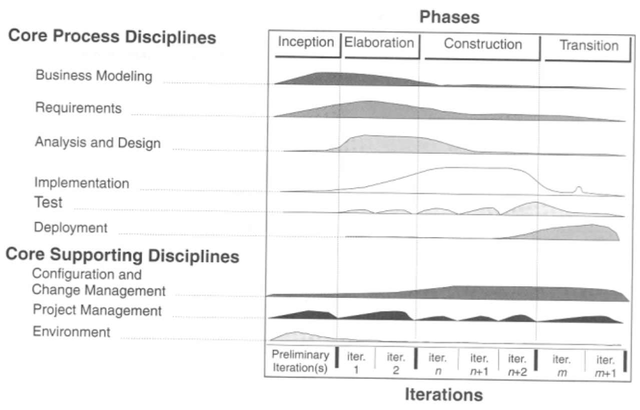
Typical LC with I&I. Quantity of iterations varies depending on the project size. Those below are called "phases":



1. Inception: The proof of concept is submitted. "Is the project feasible?". Establish the project's scope and boundary, find the critical use cases, exhibit a candidate

architecture, estimate the overall cost and schedule for the whole project, with details for the elaboration phase, and estimate risks.

2. Elaboration: Complete project plan, full architecture, and full prototype is submitted. Define and validate the architecture, baseline a detailed plan for the construction phase.
3. Construction: Contract lots of people. Achieve useful versions (alpha, beta,…), achieve adequate quality.
4. Transition: Achieve final product baseline, achieve stakeholder concurrence of completeness.



The bump on transition on testing is due to a bug that affected many modules.

Stakeholders: They are defined in the project charter. Could be the customer and/or the PM and/or the business people and/or some employers and/or partners and/or funders…

Lecture 3 Project Charter, and Requirements

Project charter: It's the contract, it's a document that formally recognizes the existence of a project. Assigns what the PM can and cannot do. It defines the project goal, objectives and scope.

Goal: Must be defined crisply and clearly.

Objectives: For a specific target, must be S.M.A.R.T (Specific, Measurable, Achievable, Realistic and Time-bound).

Scope: Boundaries of the project: the does and does not list.

Software Requirements Specifications (SRS):

- Is a description of a software system to be developed, what it is supposed to do.
- It lays out functional and non-functional requirements, and it may include a set of use cases that describe user interactions that the software must provide to the user for perfect interaction.
- At the end of the project, success will be measured, among other things, by how well the SRS has been satisfied.

Lecture 4 Work Breakdown Structure (WBS)

- A hierarchical list of work activities needed to complete the project.
- Divide and conquer mentality.
- It identifies activities at a level useful for selecting the team with the proper skills.

- Schedule, cost and duration can be estimated from it.
- Don’t plan in more detail that you can manage.
- Stop at a level where there is sufficient information for the people who will work on the activity.
- It must cover all activities (avoid too much detail).
- It’s crucial to capture all stages (requirements, analysis, design, implementation, testing).

Milestone: A significant event. Usually associated with an interim deliverable.

Activity: The smallest component of WBS, usually contains many tasks.

Task: The lowest level of effort, typically achievable by one individual.

Work package: Lowest identifiable activities to be completed.

One possible way to tackle WBS:

1. Level 1: workflows
2. Level 2: phases
3. Level 3: activities
    - A Management
        - AA Inception phase management
            - AAA Business case development
            - Elaboration phase release specifications
            - …
        - AB
        - …
    - B
    - …

Who do the breakdown structure: Anyone, a combination, similar to the idea of Project Charter. Ideally, by the people who are going to do the work.

Lecture 5 Software Size Estimation

Why is estimating poor for software:

- Lack of well-known requirements.
- Lack of prior experience.
- Hard to guess size before it’s built or at least designed.
- Management pressure.
- Developers are inherently optimistic.
- Developers ego.

Benefits of LOC:

- Easy to get.
- Easy to compare with other projects.
- Easy to track changes.
- You can know where effort is going, as if less LOC are being produce, more resources may be going to research.

- Widely used.

Prerequisite for LOC: WBS.

Beta Distribution Measurement: Requires experts. It gives a more confident estimative. Ask developers for the following LOC values: Optimistic (O), Pessimistic (P) and realistic (R). Formula:

LOC = (O+P+4R) / 6

Blitz: Based on historical observation of previous projects.

Wideband Delphi: Based on the experience of developers. Steps:

1. Gather experts.
2. Conduct group discussion.
3. Gather estimates.
4. Distribute estimates.
5. Repeat until consensus is reached.

Pros:

- Takes advantage of experience.
- Educates developers about the software.
- Reinforces Team concept.

Cons:

- Reaching consensus on an incorrect estimate.
- Can develop a false sense of confidence.
- Consensus may not be reached at all.

International Function Point Users Group (IFPUG).

Function points:

- A measure of the functionality of a given software.
- Not very suitable for algorithmically intensive apps.
- Count the number of functions in each category.
- Categories are: number of outputs, inputs, inquiry outputs, inquiry inputs, files, interfaces.
- Sum of all weighted categories is the raw function points (FP).
- Sum of all environmental factors gives the environmental influence factor (N).
- Environmental factors: Data communications, distributed computing, reusability, ease of operation…
- CAF = Complexity adjustment factor.
- AFP = Adjusted function points.
- AFP = CAF * raw FP.
- **Disadvantages:**
  - Cannot be done at the very beginning.
  - Grading from 0 to 5 is too subjective.
  - Hard to define simple, average and complex.

Feature points:

- Extends FP for other types of computationally complicated applications.
- Adds the category "number of algorithms", having a weighting factor of 3.
- Uses average weighting factors instead of simple, average, and complex values.
- Uses only 2 environmental factors: logical complexity, and data complexity.

Object points:

- Suites object-oriented software.
- Uses different weighting factors and tables.
- Very similar to FP except that objects, instead of functions, are being counted.

Mark II FP:

- Takes into account the complexity of data-rich business application software.
- Used for management information software (MIS).
- Can be applied in early stages of a life cycle.
- Software requirements are expressed in terms of logical transactions (LTs), which are the lowest level business processes. LTs are proportional to the number of data element types (DETs).
- Adjusted Mark II FP is computed by taking technical complexity factors into considerations.

Nesma: Difference with OFPUG method is that this method gives more concrete guidelines, hints and examples.

Full Function Points (FFP): Used the IFPUG rules for business application software and added extra components for sizing real-time software.

All of the methods already mentioned are called 'first generation' methods as they trace their roots back to Allan Albrecht's original ideas.

COSMIC-FFP:

- Second generation of estimation methods.
- Focused on the user functional view.
- Based on the best features of existing IFPUG, Mark II, and FFP methods.
- Standardized method that measures the functional size of real-time and infrastructure software as well as business software and hybrids of all these types.
- Compatible with modern software engineering concepts.
- The result of measurement in COSMIC-FPP is the number of data movements, i.e. the number of sub-processes, taken to represent size.

Early size estimation methods:

- **Jones Very Early Size Predictor:** Used prior to software requirements phase is complete. Utilizes taxonomy for defining software projects in terms of 'scope', 'class' and type. Cannot be used for large projects as highest assignments for scope, calss and type will give max 7675 FPs.
- **Early Function Point Analysis (EFPA):** Makes use of both analogical and analytical classification of functionalities. It's based upon the identification of software objects, like functionalities and data at different levels of detail. Estimator can have a deeper

knowledge of a specific application branch if it has already been realized more than once in a similar way, while having no or little knowledge about another branch.
- **Early and Quick Cosmic FFP:** Identify a sample of other pieces of software with similar characteristics to the new piece.

Lecture 6 Effort, Duration, Scheduling, and Fast Tracking

Size is required to estimate effort (staff-months, staff-days…) and duration (days, years…).

Effort estimation methods:

- COCOMO (Constructive Cost Model), COCOMO II and SLIM (Software Lifecycle Management)
- These methods use historical data for estimation.

COCOMO modes:

- **Organic:** Little innovation, few deadlines and constraints, stable environment; e.g. payroll, inventory.
- **Semiattached:** Some innovation, moderate deadlines and constraints, environment somewhat fluid; e.g. compilers, editors.
- **Embedded:** Much innovation, tight deadlines and constraints, complex environment; e.g. real-time systems, traffic control, weapon systems.

COCOMO levels:

- **Basic:** Uses size.
- **Intermediate:** Uses size and 15 additional variables called "cost driver"s.
- **Detailed:** Uses intermediate plus phase-sensitive effort multipliers.

COCOMO cost-drivers: The cost-drivers reflect:

- Product, hardware, personnel and project characteristics.
- Product characteristics such as the required system reliability and product complexity.
- Computer characteristics such as execution time or memory constraints. These are constraints imposed on the software by hardware platform.
- Personnel characteristics such as programming language skills that take the experience and capabilities of the people working on the project into account.
- Project characteristics of the software development project such as the IDE that is available and the development schedule.
- Each of the 15 attributes receives a rating on a six-point scale that ranges from "very low" to "extra high" (in importance or value). An effort multiplier from the table below applies to the rating. The product of all effort multipliers results in an effort adjustment factor (EAF). Typical values for EAF range from 0.9 to 1.4.

COCOMO:

- **Advantages:**
  - Historical data based on real projects.
  - Repeatable.
  - Organizations can tailor the process using the adjustment factors.
  - Relatively easy to use.

- **Disadvantages:**
  - Based on projects from 70s that followed the waterfall model.
  - It only considers KLOC to estimate effort.

Differences COCOMO and COCOMO II:

- **COCOMO:**
  1. Uses: Lines of code.
  2. Cost drivers: 15.
  3. Model: Waterfall.
  4. Data points: 63 projects referred from 70s.
- **COCOMO II:**
  1. Uses: object points, function points and lines of codes.
  2. Cost drivers: 17.
  3. Model: Spiral.
  4. Data points: 161 projects referred from 90s.
  5. Takes the CMM levels into account.

Cost estimation methods:

- Algorithmic.
- Analogy.
- Artificial Intelligence.
- Simulation.
- Expert-judgment.
- Composite.

Guidelines for cost estimation:

1. Don't mix estimation, planning and bidding.
2. Combine estimation methods.
3. Ask for justification. Unacceptable justifications:
   - I believe the effort will be 450 hours.
   - The client will not accept a higher cost.
   - Output from non-proven estimation model.
4. Select experts with experience from similar projects.
5. Accept and assess the uncertainty of effort usage.
6. Provide learning opportunities.
7. Consider postponing or avoid effort estimation.

The most influential factor in having good estimates: Requirements.

Scheduling: Assigns actual values to activities' start and finish times, leading to milestones and deliverables being determined. Created based on WBS, estimates, resources, and budget.

Two types of dependency:

- Activity dependency:
  - Coding should be done after Design.
- Resource dependency:
  - There are only 2 test machines, for 4 tests.
  - Developer X is on vacation for 1 week.

What time values should we assign to each activity: Use raw estimate! The psyched estimate, which includes uncertainties, multitasking overhead… causes the student syndrome. Therefore, use the raw estimate and add uncertainty buffers at the appropriate positions in the plan.

Gantt Chart: Assigns activity bars to each task, proportional in length to the duration of the task. Modern tools support dependencies in Gantt Charts.

Critical Path Method (CPM):

- Uses activity duration to find out the "flexibility" of each activity.
- Identifies a Critical Path (CP) which determines the duration of the whole project.
- CP = the path with no flexibility.
- Disadvantages:
  - Doesn't take resources into account, as A and B may need to be done by the same person.
  - Complexity.
  - Doesn't adapt well to making changes on the fly.

What are the most important reasons for schedule overruns?

1. Incorrect estimation of development time needed.
2. Changes in specifications.
3. Inability to detect problems early.

Schedule overruns consequences:

1. Cost.
2. Customer satisfaction.
3. Financial success.
4. Progress of dependent projects.

Recommendations to managers:

- Improve schedule estimation procedures.
- Reduce and control changes in specifications.
- Improve front-end development processes.
- Deal with the increasing complexity of software.
- Build high performance software development teams.

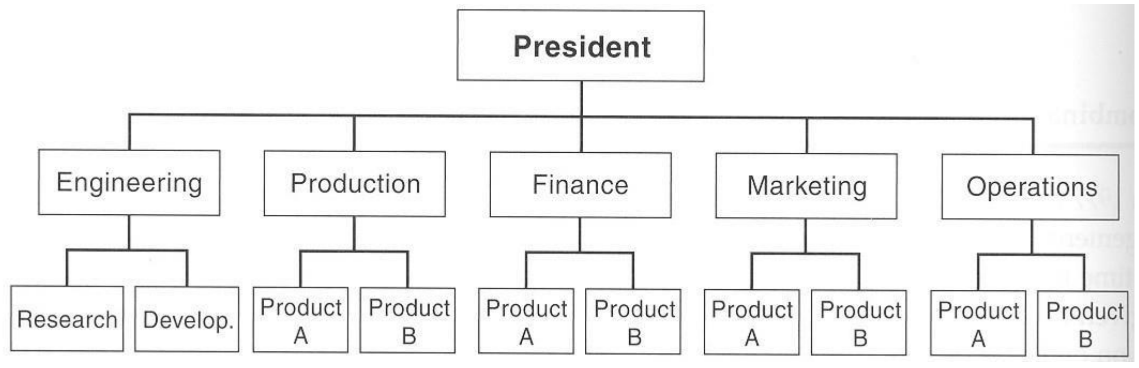
Fast Tracking Principles:

1. Calculate critical path.
2. Find the cheapest activity to crash.
3. Reduce the activity by 1 time unit.
4. Repeat step 1 until project's critical path can't be reduced (crashed) anymore.

Lecture 7 Organization, Risk Management, and Configuration Management

Organization: an interacting group of people with a unifying goal. It defines how interaction is done, by defining the responsibilities, roles and authorities.

Functional organization: Standard old-fashioned approach. People are divided into their specialties and report only to a functional manager.



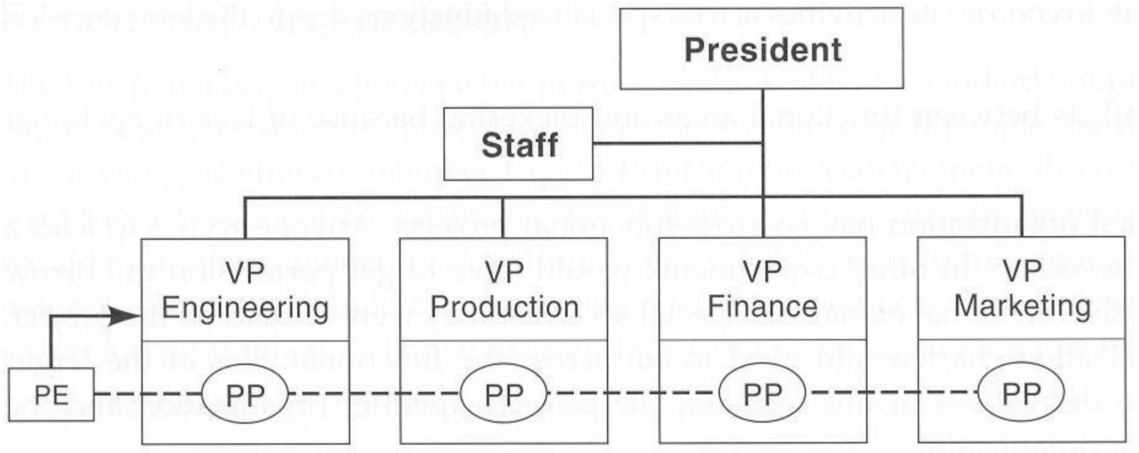
Advantages:

- Clearly defines authority.
- Eliminates duplication of functions, as, for example, all engineers are in one place.
- Encourages specialization.
- Provides career paths.

Disadvantages:

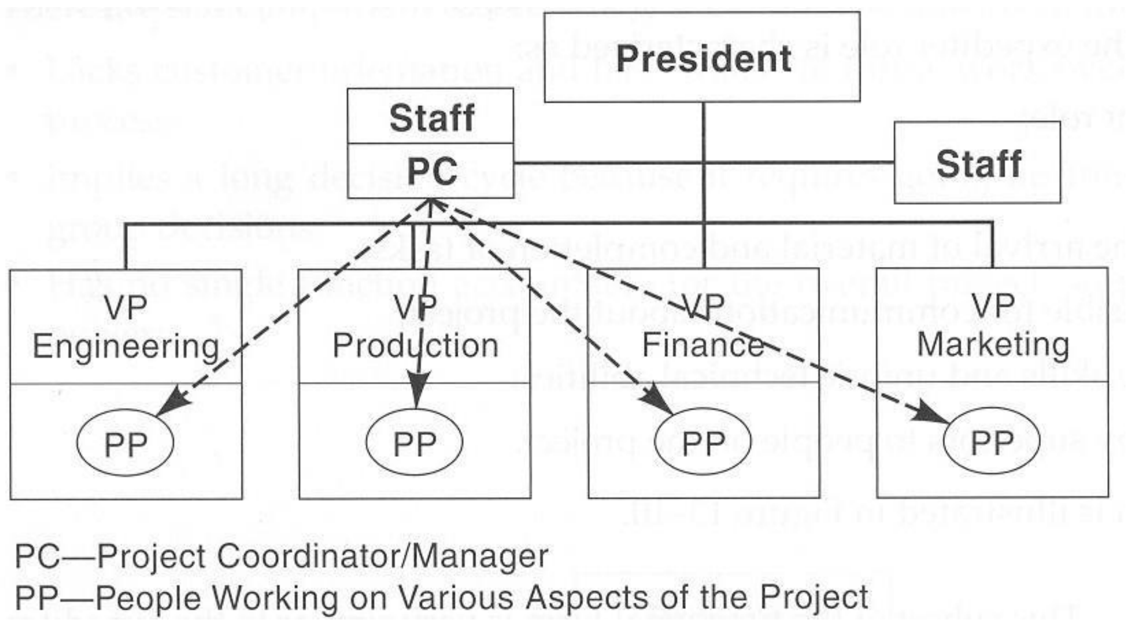
- There is no direct collaboration between the people actually working in the project (the bottom).
- If you are the costumer and needs some information, as there is no project manager, he needs to talk to the president, the president talks to the managers…

Project expeditor: Staff assistant. Responsible for overall health of the project: deliveries of material, task completion…

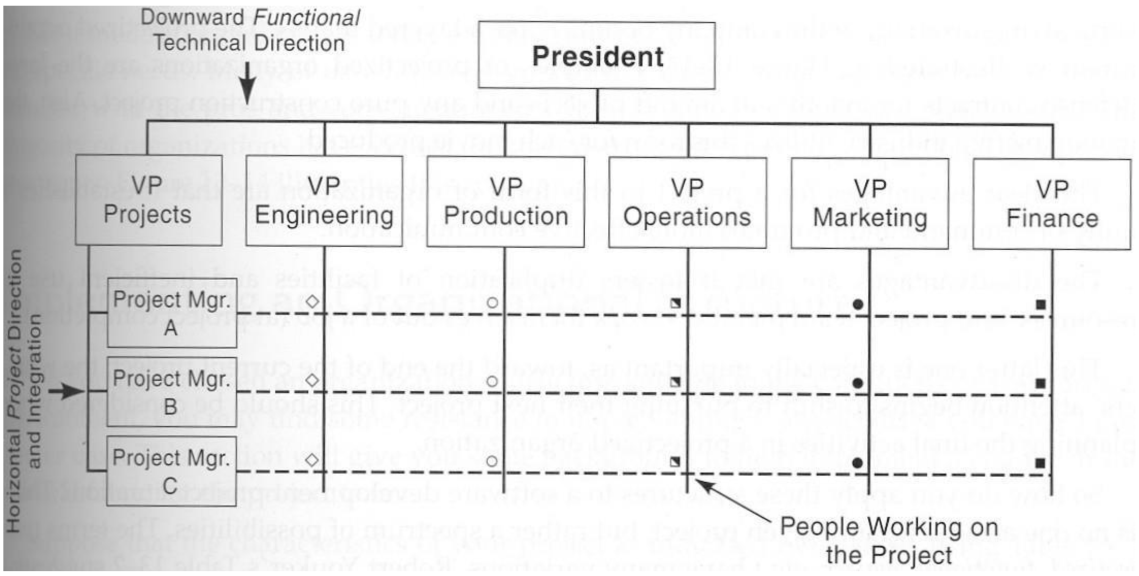


PE—Project Expeditor/Manager
PP—People Working on Various Aspects of the Project

Project coordinator: Similar to expeditor, except reports to a higher-level manager, as opposed to a function manager.


President
Staff
PC
Staff
VP Engineering
VP Production
VP Finance
VP Marketing
PP
PP
PP
PP
PC—Project Coordinator/Manager
PP—People Working on Various Aspects of the Project

Matrix: Project managers and functional managers work together.



Pros:

- Easier project integration across functions.
- Efficient use of resources.
- Enhances information flow within the organization.
- Still retains functional disciplinary teams.
- Minimizes conflict.

Cons:

- Two bosses (project managers and functional managers).
- Complex to monitor.
- Functional and project managers might disagree on priorities.
- Some duplication of effort (report to two people).

Projectized: Project-centric. PM has full authority; everyone reports to PM.

Risk management

Risk: Associated with uncertainty. It's the possibility of suffering loss. Loss could be: increased cost, delay, low-quality product…

Risk management: Is the understanding of the risks and taking them into account for project planning purposes.

Risk identification:

- The team and customer brainstorm.
- Historical data: look at previous project's risk management plans.
- Look for bottlenecks in the WBS and the network diagram.
- Types of risks: Technical, legal, political, marketing, operational…
- Sort risks based on impact and address the high-impact ones.

Mitigating risk: Once you have all risks listed in priority, develop a plan for handling them:

- Risk: unproven architecture.
- Resolution: have proven architectures standby and ready.

Controlling risk: Update the risks table on a weekly basis.

Risk visualization: Helps in giving a quick snapshot of risks and their importance. Examples: Affinity Diagram and Quadrant Mapping.

DON'T DO:

- It's too early to think about risks. We need to do:
  - Finalize the requirements.
  - Get some code running right away.
  - Put on a demo for the customers.
- Unwillingness to admit risks exist:
  - Gives the impression that you don't know exactly what you're doing.
  - Gives the impression that your boss and/or customers don't know exactly what they're doing.
- Tendency to postpone the hard parts:
  - Maybe they'll go away.
  - Maybe they'll get easier, once we do the easy parts.

Configuration management: Needed because change is inevitable. A set of activities designed to control change by:

- Identifying the work products.
- Establishing relationship among them.
- Defining mechanisms for managing different versions of them.
- Controlling the imposed changes.
- Auditing and reporting changes.

Control:

- Change is not free; we would have to ask for:
  - More money and the project is going to take longer.
  - Drop some features.

Lecture 8 Verification and Validation

Objective: To find problems before the customer does.

Verification (review, static testing): Verifies, before exiting each phase or after the completion of a milestone, whether the product of a given phase satisfies the requirements set at the beginning of that phase. Everything should be reviewed.

Validation: Actual testing of the software.

Ripple effect: Without review we face a serious ripple effect, where an error spreads to various different modules overtime, which then spreads to even more modules and so.

The review process: Gather the stakeholders and assign roles:

- **Author:** Must update the work product, it is usually the person who originally constructed it.
- **Presenter:** Explains the functionality of each section of the work product.
- **Moderator:** Ensures others perform their roles to the best of their abilities.
- **Scriber:** Records all found problems, including suggestions, and assigns severity. Completes the report.
- **Reviewer:** Each member of the team.

Dynamic testing: The objective is to fix as many errors/problems as possible. Ideally, shouldn't be done by the person that developed the software (cognitive dissonance).

Debugging: The activity of finding and fixing the error which is discovered by the test.

Types of tests:

- **Acceptance:** Is the final (not always) test based on the customer's application.
- **Alpha:** When development is nearing completion.
- **Beta:** When development and testing are nearing completion.
- **Black-box**: Based on requirements and functionality, not on internal knowledge of the design and code.
- **White-box:** Based on internal logic (conditions, branches, …).
- **Integration:** Combining all components.
- **Regression:** Retesting after fixes and patches.
- **Stress:** To test load and performance.
- **Unite:** An invokable section of code for testing.

Test plan: PM's responsibility is to ensure:

- Major test phases have been identified correctly.
- Main functions (specially the riskier ones) are tested first.
- Test plan maps well to the project plan.
- Resources are identified and available.
- All required types of tests are included (stress, regression, integration, …).

Complexity: There are three ways that complexity can be found:

1. **Edges – nodes + 2**: It doesn't count the dashed line.

2. **Count of bounded regions:** It counts the dashed line, if it doesn't exist, you have to draw it.
3. **Predicate nodes + 1:** A predicate node is a node with more than 2 exits. For each predicate note do # of exits – 1, then, at the end, sum one.

Testing paths:

1. **Statement coverage:** Execute each statement at least once as true and once as false.
   - If (A and B) print (x).
   - Test cases: A=B=True; A=B=False.
2. **Decision coverage:** Do statement coverage, plus each branch of decision is covered at lease once.
   - If (A or B) print (x) else print (y).
   - Test cases: A=B=True; A=B=False.
3. **Condition coverage:** Each condition takes on every possible value at least once.
   - If (A or B or C): print (x) else print (y).
   - Test cases: A=True, B=False, C=True; A=False, B=True, C=False.
4. **Decision/condition coverage:** Combines decision and condition coverage.
   - If (A xor B): print(x) else print (y).
   - Test cases: A=True, B=True; A=False, B=False; A=False, B=True.

Lecture 9 Software Metrics

Metric: A quantifiable measurement of software product, process, or project that is directly observed, calculated, or predicted.

Metric categories:

- **Product metrics:** describe the characteristics of the product.
- **Process metrics:** describe how to improve overall software development and maintenance.
- **Project metrics:** describe the project characteristics and execution.

Phase Containment Effectiveness: We know that the later you catch a defect, the more costly it is to fix it. Experts believe this cost to be 10 times more in a given phase compared to the previous phase. Formula:

$$\mathbf{PCE_i = E_i / (E_i + D_i)}$$

- **i**: life cicle phase: inception, elaboration, construction, transition.
- **E_i**: errors introduced and caught in phase i.
- **D_i**: errors introduced in phase I and caught in phase i+1 or later.

Metrics:

- **Modularity:** A measure of breakage localization. B/N, the lower the better.
- **Adaptability:** A measure of the ease of change. E/N, the lower the better.
- **Maturity:** A measure of the trustworthiness of the software, with trust increasing through extended usage. UT/(C0+C1), the larger the better.
- **Maintainability:** A measure of the required productivity needed for maintenance. (scrap ratio) / (rework ratio), the lower the better.

Parameters:

- **Critical Defects (C0):** Number of reworks due to errors. (Given by defects).
- **Normal Defects (C1):** Number of reworks due to low quality work. (Given by defects).
- **Improvements (C2):** Number of reworks due to going for better quality. (Given by defects).
- **New Features (C3):** Customer change requests. (Given by features).
- **Number of Cs (N):** C0 + C1 + C2. Doesn't include C3, because C3 indicates new work, not rework. (Given by defects).
- **B =** Cumulative broken SLOC due to N. (Given by broken SLOC).
- **E =** Cumulative effort spent fixing N. (Given by effort to fix (like man-days)).
- **UT** = Usage time. (Given by hours).

Lecture 10 Project Tracking, and Post Performance Analysis

A PM must be able to answer:

- How much time, cost and features have already been done.
- How much time, cost and features sill need to be done.

Main control components:

- **Scope:** Mainly controlled through Scope Management, which ensures that the software does not cross its defined boundaries.
- **Quality control:** Verification and validation of new changes.
- **Risk monitoring:** Maintain and update the top 10 list of risks, at least on a weekly basis.
- **Cost monitoring:** Using accounting data, PM can create a cost baseline, which is the amount of money that the project is predicted to cost.
- **Schedule monitoring:** Developers give the PM new estimates periodically. Estimates should be in the form "time spent, time remaining" instead of percentage completed.

Earned Value Management (EVM): A technique to measure real progress, not just the effort spent. Three major components, (partial work is not counted on those):

- **Budget Cost of Work Schedule**: What was the cost scheduled?
- **Budget Cost of Work Performed**: How much work was really done?
- **Actual Cost of Work Performed**: How much it really costed?

Post-Performance Analysis (PPA): Measures and collects the metrics to devise strategies for improving the development process. May be done during the project (iteration PPA) and/or at the end of the project (project PPA). We shouldn't wait until the very end to measure the process and collect the metrics because we may end forgetting what those metrics were. Avoid finger-pointing when collecting those during the meetings.

Lecture 11 eXtreme Programming Background

Some of the extreme practices (you have to follow it strictly):

- **Writing unit tests is done before coding**. Just enough code is written to run the tests.
- **Pair programming**, two people for one computer, pairs change once in a while.
  - Advantages: Code is being reviewed by at least to people.

    - Disadvantages: People's personality may not work together well. Efficiency is lower at the beginning for people not used to it. You may also have odd people, so one would have to work alone.
- **Refactoring** (write more readable code), refactor it soon. If the refactoring introduces bugs you can easily be caught be the extreme programming unit testing. Don't refactor if the code already has bugs, you can't make it simple or is the only thing stopping shipment.
- **Simple design,** don't design everything at once, and don't think things won't change. Simplest design: runs all test cases, contains no duplicate code, contains the fewest possible classes and methods.
- **Continuous integration**, you do integration after *every* change. Integrate one pair at a time (as you can have maximum 12 people on XP, integration in this way is possible).
- **On-site costumer**, a customer (that has the authority to makes decisions) needs to be on site to clarify things.
- **40-hour week**, avoid doing overtime, as productive goes down after a few days of overtime.
- **Coding standards**, agree on a *simple* standard as a team, and stick with it. It makes the code easier to read.
- **User stories (requirements)**, the customer writes a story. A user story is a very high-level definition of a requirement, containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it.

Idea behind XP: They accommodate change well and have less disagreement with the customer. They *don't* claim to be cheaper, faster or have less overhead. They may look fast because they release often (max 4 weeks to a release). They don't use COCOMO, LOC…

User stories: The developers help the customer to come up with the story, but it's the *costumer* that writes it. The priority is also given by the customer. Developers calculate how long it's going to take, but they *don't* commit on budget and schedule. Latter, the stories are break down into tasks.

Task: Should take a day for a pair to do it.

Tracker: A person that asks the developers about the work done and work left. Used to make decisions as dropping tasks, taking on more classes and reassigning tasks.

Stand-up meeting: Unique to XP. Shouldn't take more than 15 minutes. Used to bring up problems, not to solve them (risk management and project tracking).

Optimistic estimates: Unique to XP.

You have to release in the way that you determined it, each iteration, it may be incomplete though.

Lecture 12 Agile Software Development

Agile methodology: An abstract term instantiated by specific methodologies such as: Scrum, Crystal, Feature-Driven Development.

Agile claim: Compared to Waterfall, Agile quickly adjusts to changes.

Agile characteristics:

- Agile projects don’t commit.
- Agile projects don’t support customers that require that all the requirements are done in details. Agile operates in low to mid uncertainty.
- Agile don’t support organizations that don’t want to relinquish authority. The decision making should go to the developers.

You have to first understand the value and the principles before understating the practices.

Agile values:

1. **Individuals and interactions over processes and tools:** If the rules stop you from doing the right thing, don’t follow it. It reduces bureaucracy. People have to understand what they are doing.
2. **Working software over comprehensive documentation:** There is some documentation, but the discussion is mainly done over the working software itself.
3. **Customer collaboration over contract negotiation:** The team has to understand the customer business. Avoid CYA (cover your ass).
4. **Responding to change over following a plan:** Agile shouldn’t have *fixed* plan. Welcome the changes.

Agile principles:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software (that adds value to the customer’s business).
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress (not KLOC for example).
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely (Agile doesn’t like developers to work overtime).
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity - the art of maximizing the amount of work not done - is essential (20-80% rule).
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly (not useful for beginners).

Lecture 13 Scrum

Scrum: An agile process framework. It’s designed for teams from 3 to 9 people. The team break down their work into goals that can be completed within timeboxed iterations, called Sprints, in no more than one month and most commonly two weeks, then track progress and re-plan in 15-minute meetings (usually done in the morning) called Daily Scrums (to track progress and bring up problems). Introduces the idea of Scrum Master: the person responsible for ensuring the project is following the values, principles, and practices of Scrum (that's not a PM). Also has a Product Owner (customer representative), which is (usually) a person from your company (unlike XP), he is responsible to continuously communicate with the customer and answer the developer's questions.

Pre-game Phase: Consists of planning and architecture:

- **Planning:** Elaboration of the *Product Backlog List*, containing all the known requirements, can take several days to build initially. The requirements get *prioritized* and *estimated*. Estimates *aren't* commitments. *Usually* (unlike XP), user stories are used. The estimation and prioritization are done by the team (unlike XP). Estimation, prioritization and backlog's creation are not defined by Scrum. That is (unlike XP), you can decide how to do those.
- **Architecture:** The high-level design of the system based on current items in the backlog.

Game Phase: A.k.a. Development phase.

Rather than considering requirements, analysis, design, implementation, testing at the beginning (like Waterfall) it is done by using Sprints. The development is broken down into a number of Sprints (iterations). At the end of each Sprint, there is a release. Each Sprint begins with a *Sprint Planning* session.

Sprint Planning: Typically, a 1-day event, done in two parts: Scope, and Plan:

- **Scope:** Customers, users and management meet to decide scope. The team picks enough items from the Product Backlog List to be done during one Sprint, this will be the Sprint Backlog. It also considers personnel availability: vacations, sickness…
- **Plan:** Discuss how the work will be done. Break down the user stories into tasks that doesn't take longer than one day to be completed, put them into the "to do" task board column.

Post-game phase: Occurs when all the requirements have been met, and no more items can be invented. Signals the closure of the release.

When *not* to use Scrum:

- Large teams (more than 9 people).
- Teams with complex organizational structures (the teams must be autonomous and self-responsible).
- Geographically-scattered teams (not such a problem).
- Life or mission-critical applications (not such a problem).

Scrum Practices:

- Product Backlog List.
- Effort Estimation: An iterative process. Repeated when more information is available for some Backlog items.
- Sprint: a procedure to adapt to changes.
    - Sprint Planning Meeting.
    - Spring Backlog.
    - Daily Scrum Meeting.
- Product Owner.

- Pigs versus Chickens: In Scrum everyone is a pig (fully commitment).

Project Tracking: Done during the Daily Scrum meetings. The team discover problems, assign their own next tasks, update the Sprint Backlog, sometimes moving work, that was badly estimated. to the next Sprint. There is no CYA. During a Daily Scrum meeting, each person talks about what they did in the previous 24 hours, what are they plan to do in the next 24 hours and what are the problems that they are facing. A better way is to go over backlog items, not people, in order of priority.

Sprint Burndown Charts: Shows the amount of work remaining across date.

The following slides are from case 3, so they are not always true. Therefore, the notes below, are a summary of the real things (just follow it):

Does scrum work: It's better to ask "Did Scrum added the most value to the costumer's product?" rather than "Did Scrum improve efficiency?"

Scrum Benefits:

- Feel productive on a daily basis.
- Minimizes unproductive meeting.
- Promotes team bonding (pigs).
- License to tell people to go away (no one can stop a Sprint, except the project owner).

Scrum Shortcomings (Disadvantages):

- Lack of individual ownership can lead to disorganized design and low quality.
- Don't have time for other activities (e.g. personal development). That is, fully commitment to the project. But still has the rule of not working more than forty hours a week.
- Daily meetings can make team members nervous. That's why is better to focus on talking about the task, not the person.

Some Thoughts:

- Can run autonomously by teams within a larger organization, without the entire organization practicing Scrum.
- First Sprint(s) often focuses on design architecture, development environment…
- Last Sprint(s) is often dedicated for bug-fixing.

Tailoring Scrum:

- Scrum is not a Bible to follow, do what makes sense.
- The goal is to deliver the most value to the customer's product, not just follow a set of rules.

Lecture 14 Crystal, FDD and DSDM

Crystal is a *family of methodologies*: You need to treat the project depending on the criticality of the software.

Crystal idea: *Follow* certain rules. Then, after knowing those rules, *break* those rules. Finally *leave* those rules and create your own rules.

Unique to Crystal:

- **Frequent Delivery:** Unique to Crystal, you don't just release frequently, you deliver frequently. You integrate your software into your costumer's product.
- **Osmotic Communication:** You must be in an open space where you can listen everyone else's conversation. So that you can hear it, and sometimes listen to then, so that you may help them.

Common Rules:

- Incremental development cycles of no more than 4 months; 1 to 3 months recommended. Crystal's teams can have up to 80 people.
- You can use Scrum practices or any other tools, methodologies.

Activities slides are not going to be asked.

FDD (it's an Agile methodology) Roles and Responsibilities:

- It's the only one that defines very specific roles.

FDD Practices:

- FDD works only ins one feature.

Dynamic Systems Development Method (DSDM) (it's an Agile methodology):

Differences between it and other Agile methodologies

- Focused on the business side. That is, focus on the best return of investment (law of diminished return).
- Customer plus the developer can do whatever they want. Managers don't have authority.

He said to ignore the slide "How is this different from XP?"

- Timeboxing: The same as XP, you have to release when you said you would release, even though it's not complete. Only XP and DSDM are very specific about it.
- Configuration management is big in DSDM

What every development methodology has to have:

- Incremental and iterative.
- Short iterations (relative to the size of the team).
- Static testing (reviews) in addition to dynamic testing.
- Architecture first.
- Prototype.
- Release often.
- Control change (accommodate change).
- Track, track, track!
- Adjust the plan as needed.
- Customer involvement.
- People factor.

Lecture 15:

ISO 9001 is not so important to software anymore.

ISO 90003 is specific for software. It tell how to interpret 9001 for software specifically.

Comparing CMM with ISO 9001:

- CMM is not binary. That is, instead of having two states (ISO 9001 certified or not certified). It has 5 levels.
- There is a correlation between ISO 9001 and CMM level 2.

Comparing CMM to CMMI:

- Better documented, clear how to assess your methodology and how to tailor it.
- The main thing are those 3 steps.

IEEE standards:

- Easy to communicate with other companies.
- Can remind you what is missing in your project by reading the standard.

Lecture 16: (guest IBM)

Kaizen: Continuous small improvements.

- Epic: Many stories.
- Story: Many tasks.

- Sprint: 2 calendar weeks.
- Milestone: 4 sprints.
- Release 1-2 milestones.
- As a developer, the most import deadline is the sprint (Kaizen).

- Stand up meeting used to catch up what others are doing.

Lecture 17: (guest IBM)

Main subjects:

- Backlog grooming
- Multi-squad team: how do we work together
- Working with Offering Managers (business folks)
- Maintaining quality

Backlog grooming:

Backlog: A prioritized list of stories to be completed in a milestone/release.

Grooming: a regular sprint (mid-milestone) to refine the backlog.

Backlog grooming: Finds the epics/stories that have highest priority for next milestone.

Sprint planning: It’s taken out from the backlog.

Coordinating Multi-Squad Teams:

Features from other squads sometimes overlaps with yours. The Scrum Master is responsible for managing that.

Cross-squad meeting: Once or twice a week, a stand-up among the Scrum Masters.

Design meeting: If you don't do it, you would be breaking alignment, staying true to the customer. Everyone has to agree on the design before the developers start programming.

Quality gates:

- Agile ensures acceptable quality. Automate testing when possible. Relative cost for fix bugs, based on time of detection: Requirements/Architecture 1x, Coding 5x, Integration/Component Testing 10x, System/Acceptance Testing 15x, Production/Post-release 30x.
- DevOps make sure your code goes smoothly to the production.
- Code coverage: how much of your code is being covered by your tests.
- Static analysis: Tool that informs you if you're following the correct coding pattern, like not catching a null pointer.

Defects: Divided into severities:

- Severity 1: Product is broken.
- Severity 2: Product is functional, but behaviour is broken.
- Severity 3: bad, but not a show stopper.

SCRUM and XP you have to know well

The others you must know their differences to the other methodologies.

Project Presentation:

Individual grading. You're students, not bidders in this case.

Design: Artificial intelligence, cloud

I'll probably present: WBS, Development Methodology (general I&I), Schedule and Budget, IA and ML

PPA: These we did good, these we didn't do good so in the next time we would do it like this (and say it)

Customer Demo:

You're the bidding company. The TA has already seen your project plan. It can be presented by only one person.

Documents to be delivered:

1. Project plan.
2. Software Requirements Specification

3. Software Design Document
4. Risk Management Plan
5. Configuration Management Plan
6. Verification and Validation Plan

Final:

Assignment is going to be in the final. And everything else.

One question of the industry guest lecture.

Cost divers, you don't need to know what they are, but you have to know what they do (adjust the project based on external factors).

1:50 minutes instead of 3 hours

<u>3 full-answer questions from: Class discussions or assignment</u>

There is no PM in Agile, except in "FDD?".

In Agile there is no behind, and assignment more resources will not help (if it is behind).

You don't have commitment in Agile

1) **What is the Capability Maturity Model (CMM)?**

The capability maturity model ranks companies according to their software development process. The CMM has five levels:

1) Initial: The process is chaotic an ad-hoc (executes only one task, can't be generalized), it depends on the individual effort.
2) Repeatable: There're some project management processes, like defining the schedule, budget and functionality. They try to repeat the same successful process on projects with similar characteristics.
3) Defined: The company has a standardized software process for both managements and engineering.
4) Managed: Detailed measures about the software process are collected. The software process is quantitatively understood and controlled.
5) Optimizing: Continues improvement is enabled by the quantitative measures.

2) **What are the differences between the waterfall and I&I processes? Why to use I&I in software?**

The waterfall method is not iterative, that is, it follows a continue flow. You can only move between adjacent stages. In waterfall, the iteration requirements, analysis, design, implementation and testing are done only once. There is a big effort in doing the requirements phase as correct as possible, as it will not change. Moreover, there is no continues feedback from the customer. Usually, the customer would see his product only when it was finished.

Furthermore, the waterfall model may work well enough for projects that are well known and that have been done many times before, which is not the case of software.

The iterative and incremental process allows the process have various repeated iterations. Each iteration consists of requirements, analysis, design, implementation, testing. There's a constant feedback from the customer.

The typical life cycle in I&I consists of inception, elaboration, construction and transition. Each of those may have many iterations. The inception phase is used to check if the project is feasible, it estimates the time, budget and risks of the project. By the end of the inception phase, you have a proof of concept and an initial prototype. By the end of elaboration, you have a full prototype, full architecture and a complete project plan. During construction, more people may be hired in order to build the product. By the end of the elaboration phase you'll have your first external release, a beta. By the end of the transition phase, you'll have your final product.

Waterfall: Requirements first, custom development, change avoidance, ad hoc tools.

I&I: Architecture first, component-based development, change management, round-trip engineering.

3) **What is Software Requirements Specifications (SRS)? Why do we use it?**

The Software Requirements Specification stablishes the functional and non-functional requirements of the project as well as some use cases so that the stakeholders and developers can understand it more clearly. It's a description of what the software needs to do. It's the main document which we derive both the software design specifications and validation tests. Most bugs are caused by unclear or missing requirements. Therefore, why use it so that we don't need to waste time and money correcting errors that could have been caught before.

4) **What is the Work Breakdown Structure (WBS)?**

The work breakdown structure is a hierarchical list that defines the activities to be performed during the conception of the product. It has to be specific to the point that people who will do the tasks, have sufficient information, but not overly specific to the point that it becomes too complex. The level reached should be useful for selecting a team with the proper skills needed for that activity. Based on the WBS, cost and schedule can be estimated. Similarly, to the Project Charter mentality, anyone could do the WBS, it could be the PM, the customer, business people, all the stakeholders, a combination of those… but it should be ideally be made by the people who are going to do the work. To do it, it's important to go through some existing documents, as the requirements, design documents and customer conversations.

5) **What is LOC? Why should we use it?**

LOC is the Lines of Code. The LOC can be used to estimate schedule and cost. Why should use LOC because 1) It's easier to present than Function Points; 2) It's widely used, so we can compare it with other projects; 3) It's used in many cost and schedule estimator methods; 4) It's easy to get; 5) Easy to track changes; 6) You can know where the effort is going. The pre-requisite to get the lines of code is the WBS.

6) **Cite and explain three size estimator methods:**

The beta distribution estimation requires experts, you should ask them to give LOC values optimistically, pessimistically, realistically. Then, apply the formula: LOC = (O + P + 4*R) / 6.

Blitz is based on historical observation of previous projects.

Delphi is based on developers' experience. You ask them to estimate LOC, they return their estimation in a secret way, you show the estimation to everyone (without revealing who provided it), then you debate it and repeat the process until consensus is reached. The advantages are 1) people get a better understanding of the project; 2) reinforces team concept; 3) takes advantage of experience. The disadvantages are 1) consensus may not be reached; 2) a wrong consensus may be reached; 3) can develop a false sense of confidence.

7) What are functions points? Why do we use them?

Functions points is a way to measure the size of the project. We use a table and define each of the functions as simple, average or complex. Then we use a complexity adjustment factor based on the environmental vectors of the project. But, it's hard to define as simple, average and complex the functions, as well as from 0 to 5 the environmental factor. Moreover, it can't be done at the very early. Feature points simplifies the FP. Object points suites object-oriented software, uses objects instead of functions. Mark II FP takes into account the complexity of data-rich business application software. COSMIC-FFP: Its result are sub-processes, it's compatible with modern software engineering concepts.

8) How to calculate effort, duration and scheduling?

There're methods to do so, all them requires the size of the project, like COCOMO. The COCOMO method can output the effort and duration of the project. Is has three modes: Organic, Semiattached and Embedded. It also has different levels. The second level include 15 cost drivers. The cost drivers are used to give a more precise estimation. However, it's hard and subjective to rank all those 15 cost drivers. They are ranked between very low to extra high. The cost drivers reflect the characteristics of the product, personnel, project and computer. Disadvantage of COCOMO includes considering only KLOC to estimate effort, also, from 70s projects that used waterfall.

9) How to estimate cost?

There are many ways to estimate cost, like algorithmically, analogically, using AI, simulation, expert-judgment and composite. Most important factor in having good estimate are requirements.

10) What are the disadvantages of critical path?

It doesn't take into account that the same person may be the responsible for two different activities, that is, doesn't take resources into account. It's complex and can't adapt well to changes on the fly.

11) Fast tracking principle:

- Find the CP.
- Find the cheapest element.
- Crash that element.
- Repeat.

12) Talk about configuration managements, highlighting its pros and cons:

Functional: Old-fashioned. Pros: clearly defines authority; encourages specialization. Cons: No direct communication among the people at the bottom; hard to communicate with the client, has to go all the way down then all the way up.

Project expeditor: Person that can help the overall health of the project, can communicate with people at the bottom and function manager.

Project coordinator: Similar to expeditor, but he can communicate higher-level managers.

Matrix: PM and functional managers work together. Pros: Better communication, easier project integration. Cons: Two bosses, complex to monitor.

Projectized: Focused on the project, PM has full authority.

13) **What is the importance of risk management?**

The idea is to know the risks (as much as possible), try to mitigate them, and have a strategy to act if they occur. The risk identification is a brainstorm of the team and the customer. Look at historical data as well, and WBS bottlenecks.

14) **Two methods to calculate the complexity:**

- Edges – Nodes + 2. Doesn't count the dashed line.
- Count of bounded regions. Counts the dashed line.

15) **What is validation and verification and why should we care about it?**

Validation is the actual testing of the software. Verification if the product satisfies the requirements set at the beginning of that phase. If that is not done, we would face the ripple effect, where an error spreads to many different modules, which then spreads to other modules and so on.

16) **What is software metrics and why should why care about it?**

Software metrics are useful to better manage the product, process and project. The phase containment effectiveness is related to bugs found and caught during one phase. The formula is: $PCE = E_i / (E_i + D_i)$, where *i* is the phase; E_i are the errors introduced and caught in phase *i*; D_i are the errors introduced in phase *i* and caught in phase *i* + 1.