LECTURE 1: SOFTWARE DEVELOPMENT PROCESSES

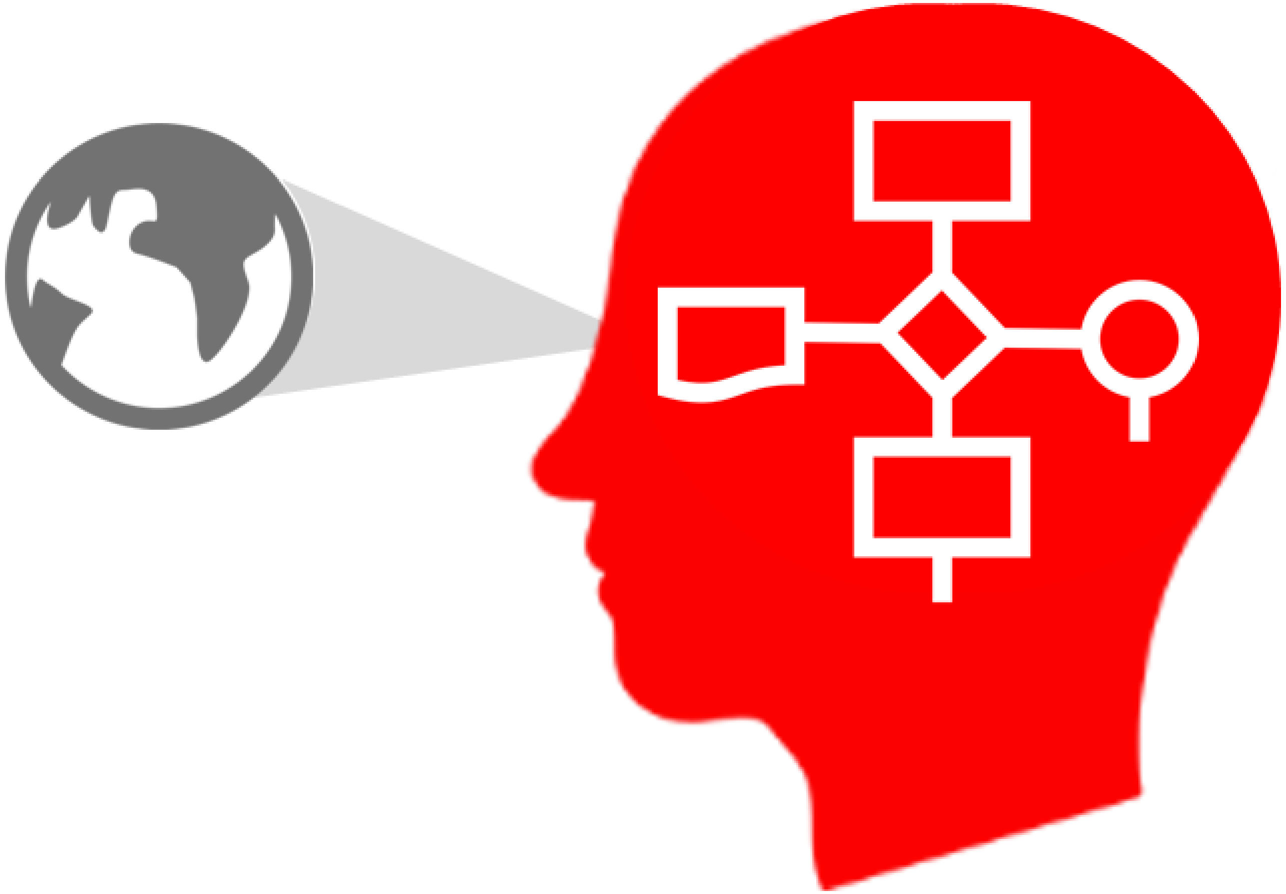
Lotfi ben Othmane

Iowa State University

Fall 2017

Slides are based on the lecturers
Given by Kent VanderVelden
in Fall 2016

Mental model



No class next week as I will be in a conference

- Replacement options
  1. Wednesday 8:00 to 9:15
  2. Wednesday 4:10 to 5:25
  3. Friday 8:00 to 9:15
  4. Friday 4:10 to 5:25

- Week Sep 18, Sep 25, Oct 2, Oct 9

TASKS OF A PROJECT MANAGER

- Knowing what the customers want
  - requirements
- Estimating development time
- Estimating required resources
- Managing change
  - employee loss, resource loss, requirement change
- Knowing when you’re done
  - predefined requirements vs. vague requirements
- Dividing the work
- Coordinating the work
- Properly estimating value
- Tracking time, resources, quality, productivity, effectiveness, etc.

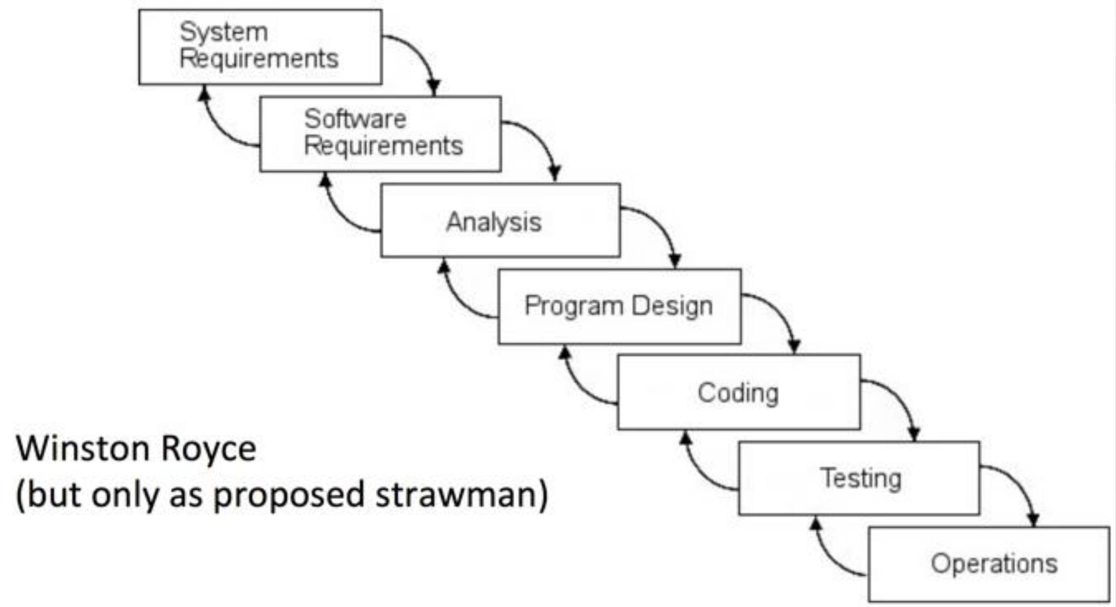
=> FIRST TOOL: DEVELOPMENT PROCESS

SOFTWARE DEVELOPMENT PROCESSES

- Process
  - A set of **artifacts**, **activities**, and **roles**
  - Criteria for progressing from one activity to another
  - **Activities** are performed by **roles** to produce **artifacts**
    - Examples
      - **Artifact**: Requirements specification
      - **Activity**: Define requirements
        - Prototype
        - Specify
        - Review
    - **Role(s)**: Systems Engineer, Architect, Voice of the Customer
    - Criterion: Requirements must be reviewed before proceeding to design phase

CHARACTERIZING PROCESSES
THE WATERFALL MODEL

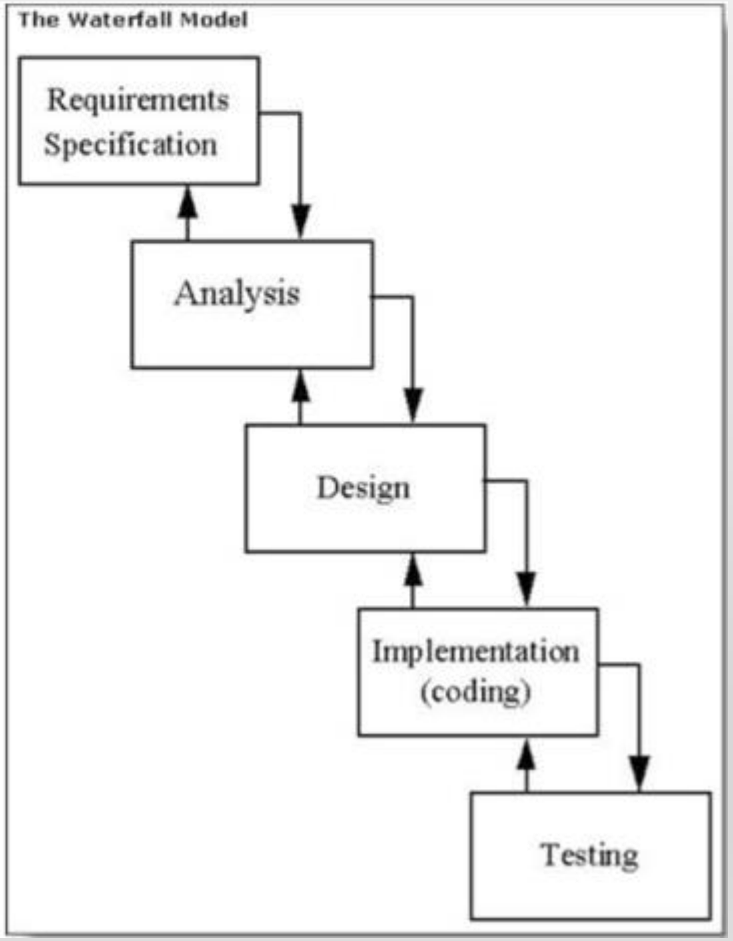
- Process viewed as a sequential set of activities
  - Elicit requirements, analyze and design, code, test, release
  - Finish one stage before moving to the next
    - Backtrack if necessary
  - Prototyping could be part of requirements determination



Winston Royce
(but only as proposed strawman)

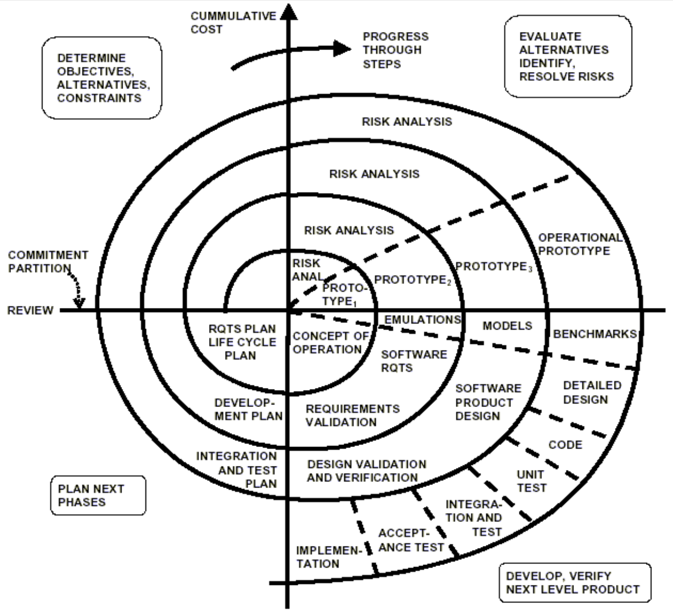
ASSUMPTIONS FOR THE WATERFALL MODEL

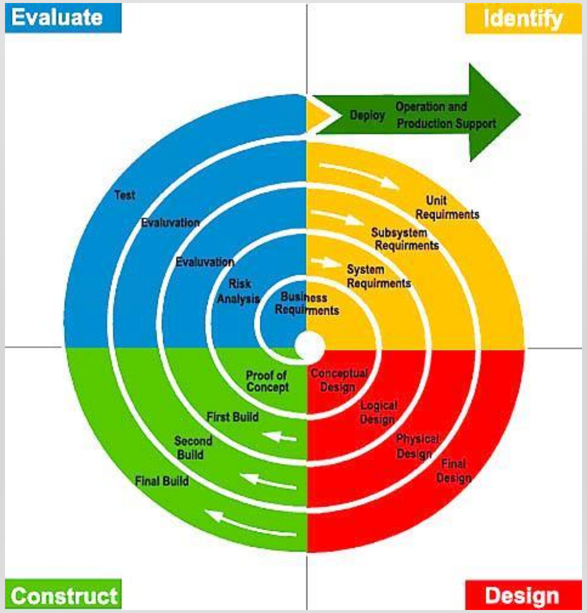
1. Requirements are understood and specified before code is designed
2. Requirements analyst produces a real written specification
   - Significant effort to develop useful specification
   - Evaluate for completeness, consistency, etc.
3. Software are built in accordance with written requirements
   - Like a checklist



CHARACTERIZING PROCESSES
THE SPIRAL MODEL

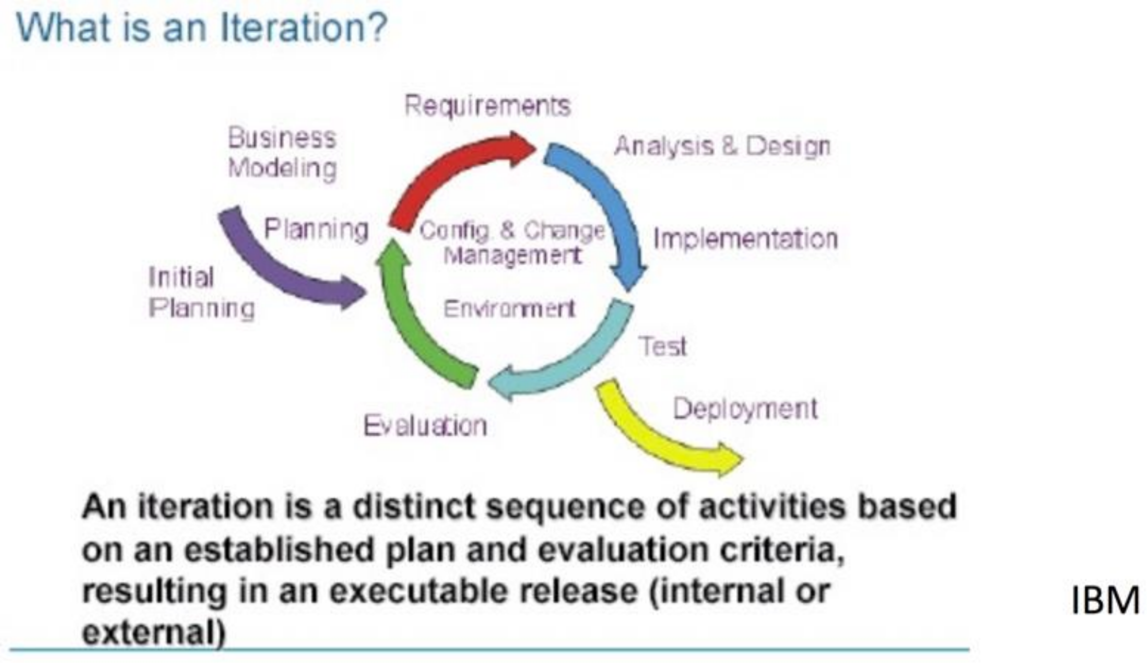
- Repeating cycles of increasing scale
  - Identify risks and values, determine next requirements, build next version by extension, increase scale
  - Early iterations may be prototypes
  - Each iteration, similar to waterfall approach





CHARACTERIZING PROCESSES
THE ITERATIVE MODEL

- Generalization of spiral model
- Process viewed as a sequence of iterations, each buildings on the last
  - Build minimal useful subset, test, release, build next version by extension. Early iterations may be prototypes



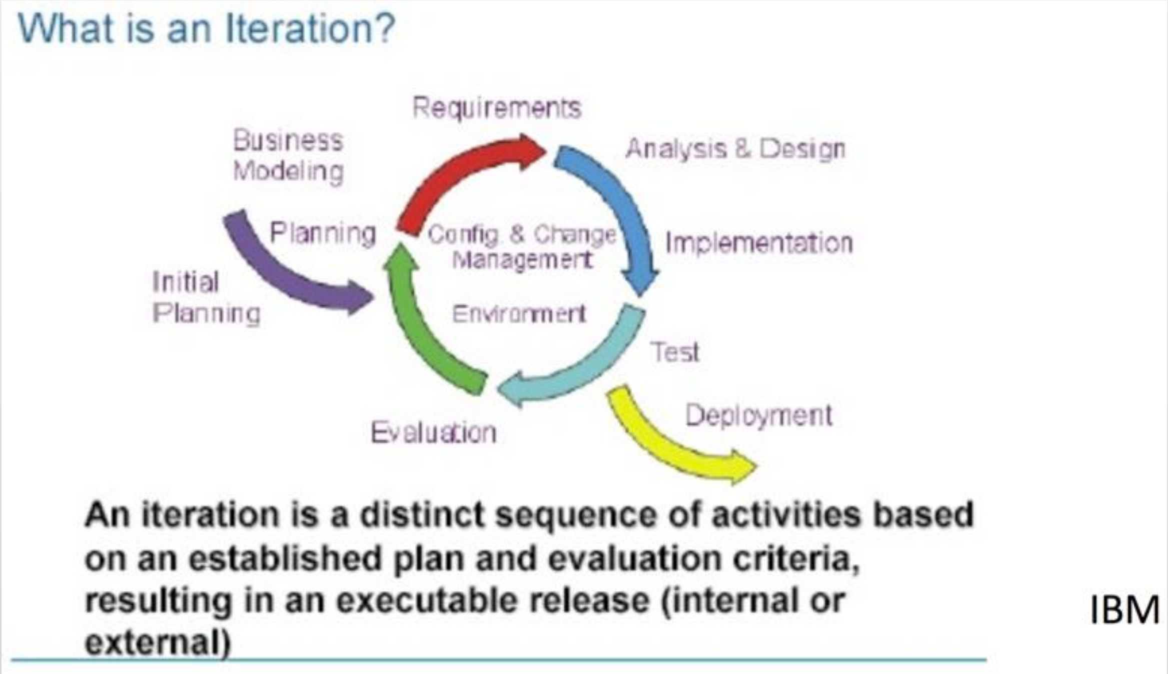
What is an Iteration?

An iteration is a distinct sequence of activities based on an established plan and evaluation criteria, resulting in an executable release (internal or external)

IBM

ASSUMPTIONS OF THE ITERATIVE MODEL

1. Requirements can be understood well enough to build a minimal useful subset
2. Early iterations allow for extension (and contraction) of subsets
   - Clearly identified model structure



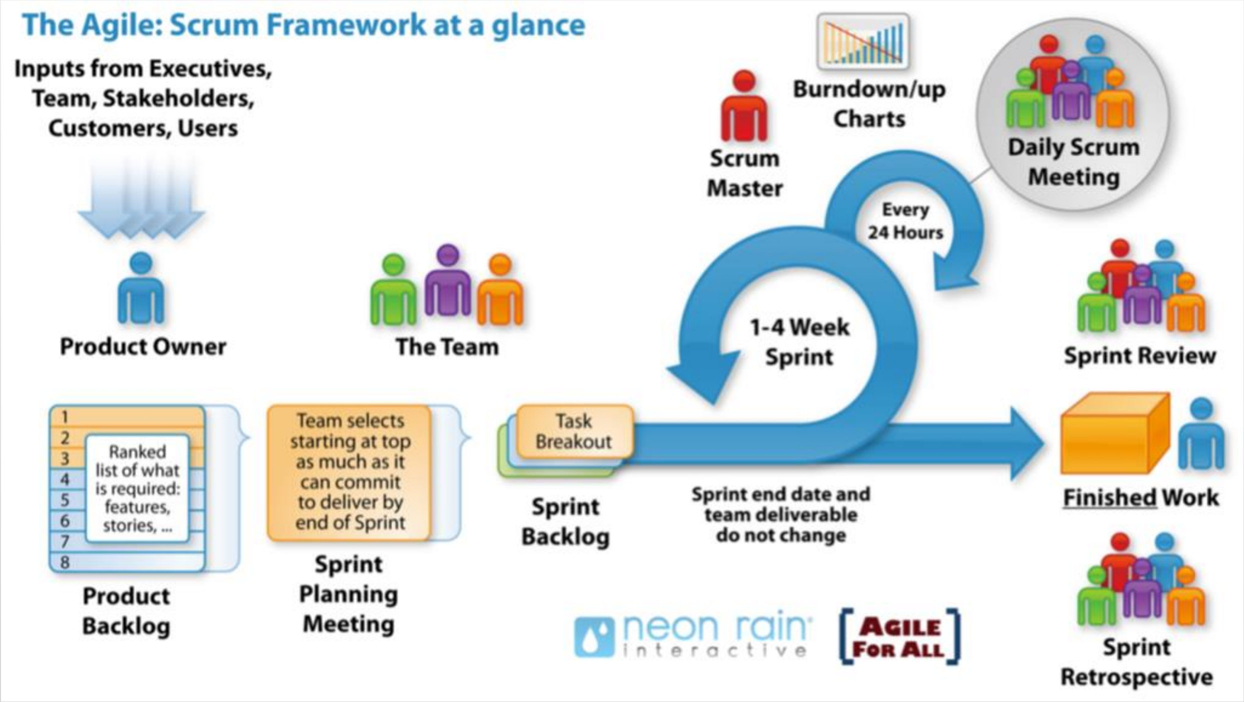
What is an Iteration?

An iteration is a distinct sequence of activities based on an established plan and evaluation criteria, resulting in an executable release (internal or external)

IBM

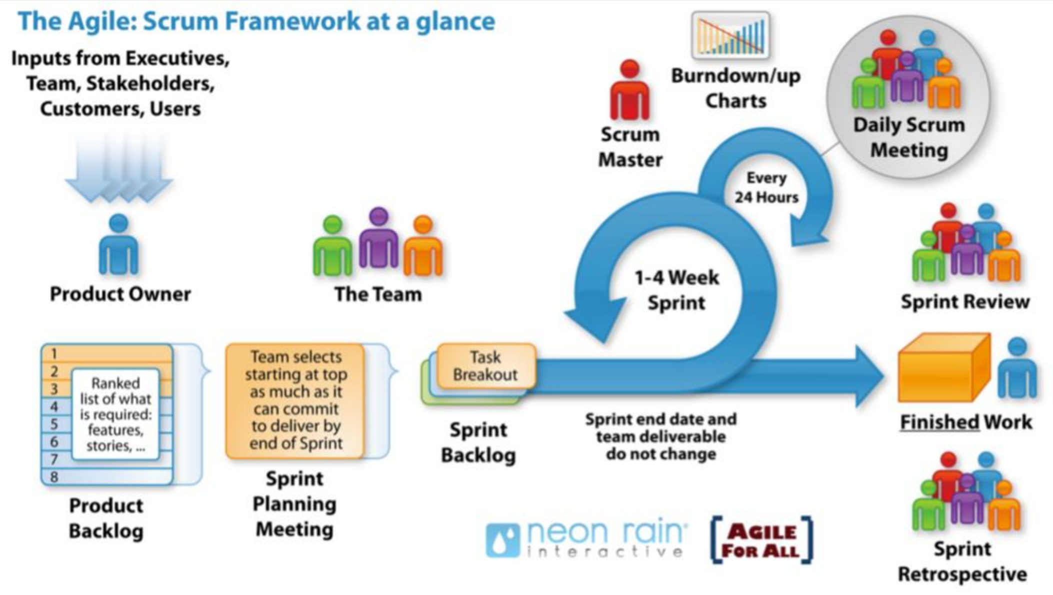
CHARACTERIZING PROCESSES
THE AGILE METHODS PROCESS

- Many small, quick iterations, known as sprints
- Each iteration implements a user story
- Client validates increment


The Agile: Scrum Framework at a glance
Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Ranked list of what is required: features, stories, ...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective
neon rain interactive
AGILE FOR ALL

ASSUMPTIONS AGILE PROCESS

1. Requirements cannot be understood before code is developed
2. Requirements gathered informally from customer
   - Code is the only record
3. Requirements can be implemented in small increments


The Agile: Scrum Framework at a glance
Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Ranked list of what is required: features, stories, ...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective
neon rain interactive
Agile For All

APPLIED PROCESSES HAVE VARIABILITY

- Continuum of specificity, time invested, and accountability
- Form of requirements, design, test plan
    - Written document
    - Knowledge in the heads of the development team
- Review procedures for documents and code
    - Formalized inspections with criteria for passing
    - Informal peer review meeting
    - Office mate reviews
- Release criteria
- Coding style
    - Enforced standard
    - Everyone has their own style
  - Project manager, systems engineer, architect, developer, tester
    - Dedicated people? Shared roles?

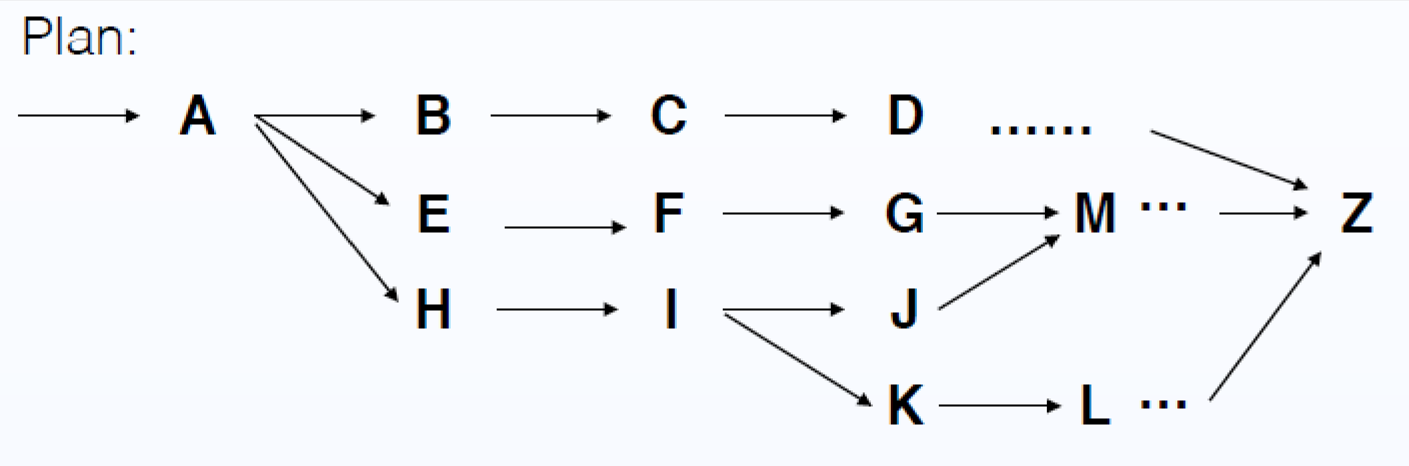
Critical projects lead to more formalized process

- Avionics, medical software, defense

EXPECTATION VS. REALITY
RATIONAL VS. IRRATIONAL

You make decisions

Perfect forward moving process

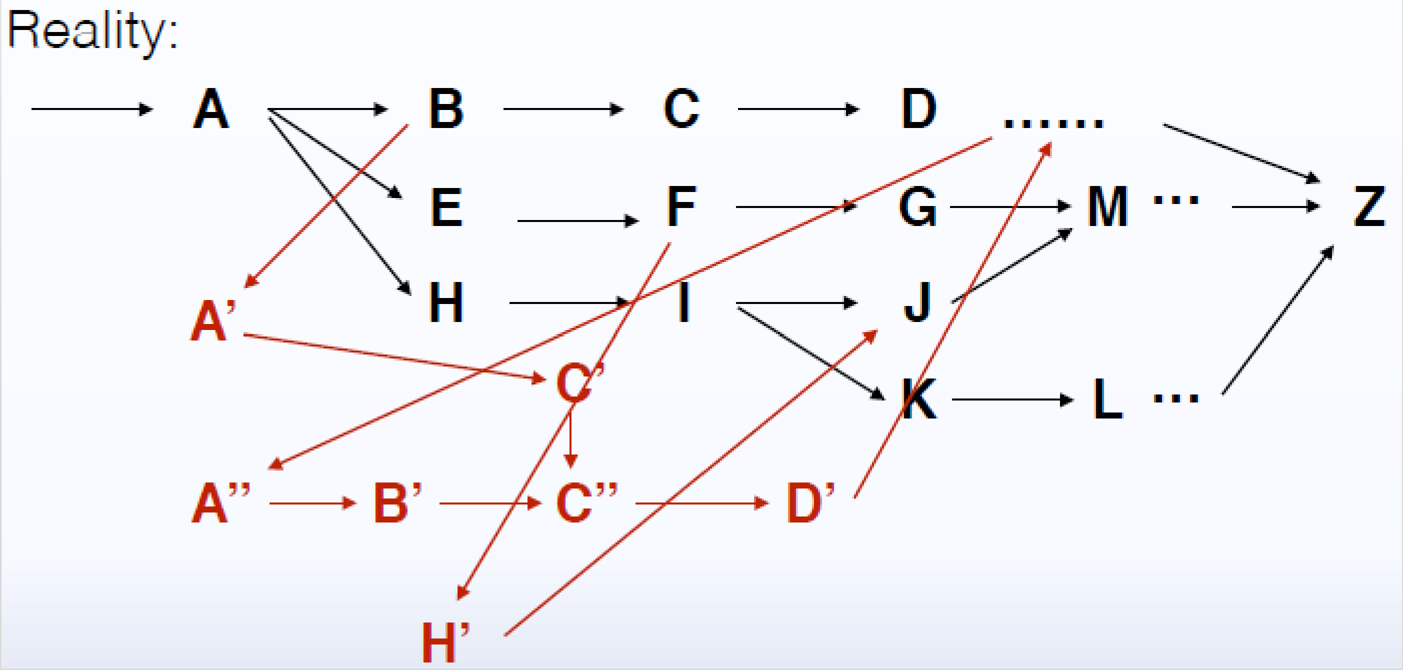


EXPECTATION VS. REALITY

RATIONAL VS. IRRATIONAL

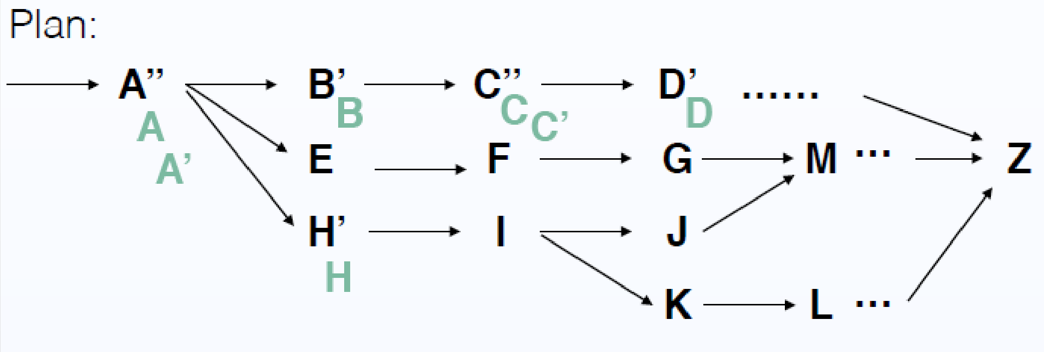
Reality is different from what is expected. Decisions may need adjustment

Restarts, redos, undoes, and lessons learned



RATIONALIZING REALITY

- Document as if it had been rational
  - Readers can follow sequential story



- Explain significant decisions
  - What alternatives are there?
  - Why was A'' chosen rather than A' ?
  - Avoid re-implementing "mistakes" during maintenance
- Later maintainers can understand the tradeoffs and be guided by them
  - Record the work, not the result

MANAGING THE DIFFICULTIES

- Software engineering provides a range of processes and methods to address these difficulties, for example:
  - Processes
    - Spiral: risk assessment, mitigation
    - Iterative, such as Agile: rapid customer feedback
  - Methods
    - Model Driven: problem modeling
    - Prototyping: early validation

- Different approaches, different assumptions

TAILORING THE PROCESS

- Match goal to environment
- Pick appropriate roles, artifacts, activities, modes of communication
- Process evolves with the project
- What motivates people?
  - Having an impact?
    - Frequency of feedback?
  - Tailoring process to improve employee
    - Projects as opportunities for new skills, tools, resources

PROCESS COMPLEXITY AND PROJECT SCALE

- On large or complex projects, process helps assure that:
  - Work assignments are properly divided and assigned
    - result in code that work together
      - Modules work together to produce the desired result
  - Teams at different sites understand the interactions among their work and the work at other sites
    - Test each others code
  - How about the following?
    - Validate that requirements are feasible
    - Validate that requirements express what the customer or market wants

SELECTION OF DEVELOPMENT PROCESSES

- Fit the process to the situation (team, project)
  - Define the appropriate artifacts, roles, activities
  - Measure the result
    - In progress
    - At project's end
  - Continually evaluate
    - How well are we doing?
    - What should we do to improve?
    - Client happy?

SELF-CHECK

- What are the common software development approaches?
- Why do we need a variety of processes?
- Why applied processes are different?
- What are the tasks of a project manager?