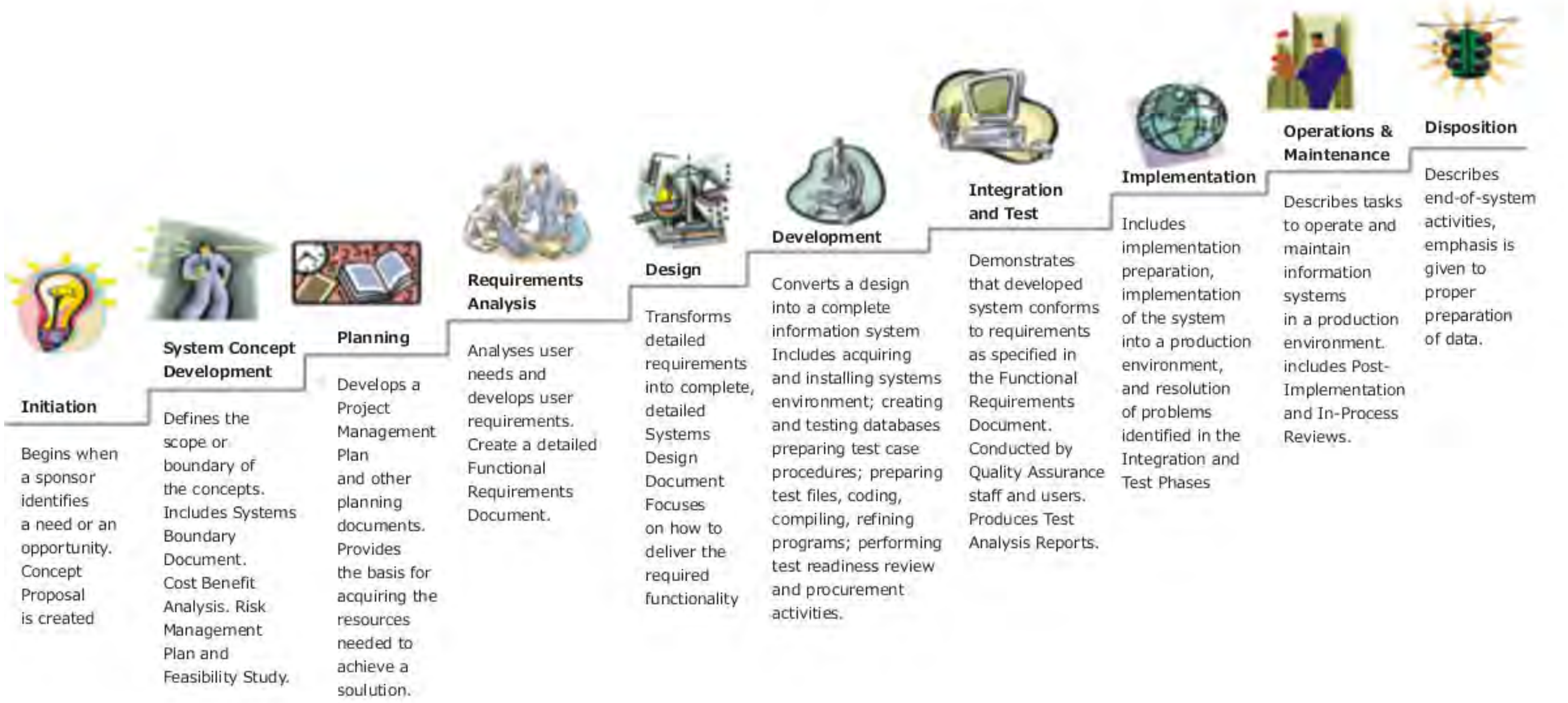
SOFTWARE PROCESS

Software Engineering

Dr. Raj Singh

SOFTWARE DEVELOPMENT PROCESS

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)



Initiation

Begins when a sponsor identifies a need or an opportunity. Concept Proposal is created

System Concept Development

Defines the scope or boundary of the concepts. Includes Systems Boundary Document. Cost Benefit Analysis. Risk Management Plan and Feasibility Study.

Planning

Develops a Project Management Plan and other planning documents. Provides the basis for acquiring the resources needed to achieve a soulution.

Requirements Analysis

Analyses user needs and develops user requirements. Create a detailed Functional Requirements Document.

Design

Transforms detailed requirements into complete, detailed Systems Design Document Focuses on how to deliver the required functionality

Development

Converts a design into a complete information system Includes acquiring and installing systems environment; creating and testing databases preparing test case procedures; preparing test files, coding, compiling, refining programs; performing test readiness review and procurement activities.

Integration and Test

Demonstrates that developed system conforms to requirements as specified in the Functional Requirements Document. Conducted by Quality Assurance staff and users. Produces Test Analysis Reports.

Implementation

Includes implementation preparation, implementation of the system into a production environment, and resolution of problems identified in the Integration and Test Phases

Operations & Maintenance

Describes tasks to operate and maintain information systems in a production environment. includes Post-Implementation and In-Process Reviews.

Disposition

Describes end-of-system activities, emphasis is given to proper preparation of data.

SOFTWARE DEVELOPMENT ACTIVITIES

Planning

An objective of each and every activity, where we want to discover things that belong to the project.

Analysis & Design

Analysis of requirements and design of software is done throughout development

Implementation

Implementation is the part of the process where software engineers actually program the code for the project.

Testing

Software testing is the process to ensure that defects are recognized as soon as possible.

Deployment

Deployment starts directly after the code is appropriately tested and approved for release to production environment.

Support

Software training and support is important, as software is only effective if it is used correctly.

Maintenance

Maintaining and enhancing software to new requirements can take substantial time and effort as missed requirements may force redesign of the software.

SOFTWARE PROCESS MODEL

A structured set of activities required to develop a software system

- Specification
- Analysis, design and implementation.
- Validation
- Evolution

A software process model is an abstract representation of a process

It presents a description of a process from some particular perspective

SOFTWARE DEVELOPMENT PROCESS

A structure imposed on the development of a software product.

A framework that is used to structure, plan, and control the process of developing an information system.

Several software development approaches have been used since the origin of information technology.

PROCESS PATTERNS

A process pattern

- describes a process-related problem that is encountered during software engineering work,
- identifies the environment in which the problem has been encountered, and
- suggests one or more proven solutions to the problem.

In more general terms

- a process pattern provides you with a template.
- a consistent method for describing problem solutions within the context of the software process.

PROCESS PATTERN TYPES

Stage patterns

- defines a problem associated with a framework activity for the process.

Task patterns

- defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice

Phase patterns

- define the sequence of framework activities that occur with the process, even when the overall flow of activities is iterative in nature.

PROCESS MODELS

SOFTWARE DEVELOPMENT MODELS

Prescriptive Models

Traditional

Agile Models

Modern

PRESCRIPTIVE MODELS

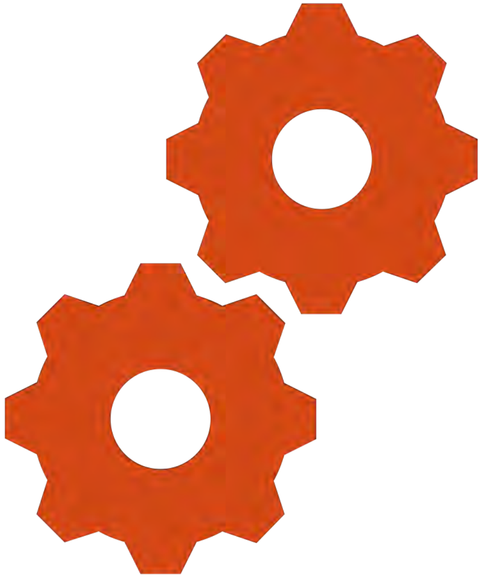
Prescriptive process models advocate an orderly approach to software engineering

That leads to a few questions …

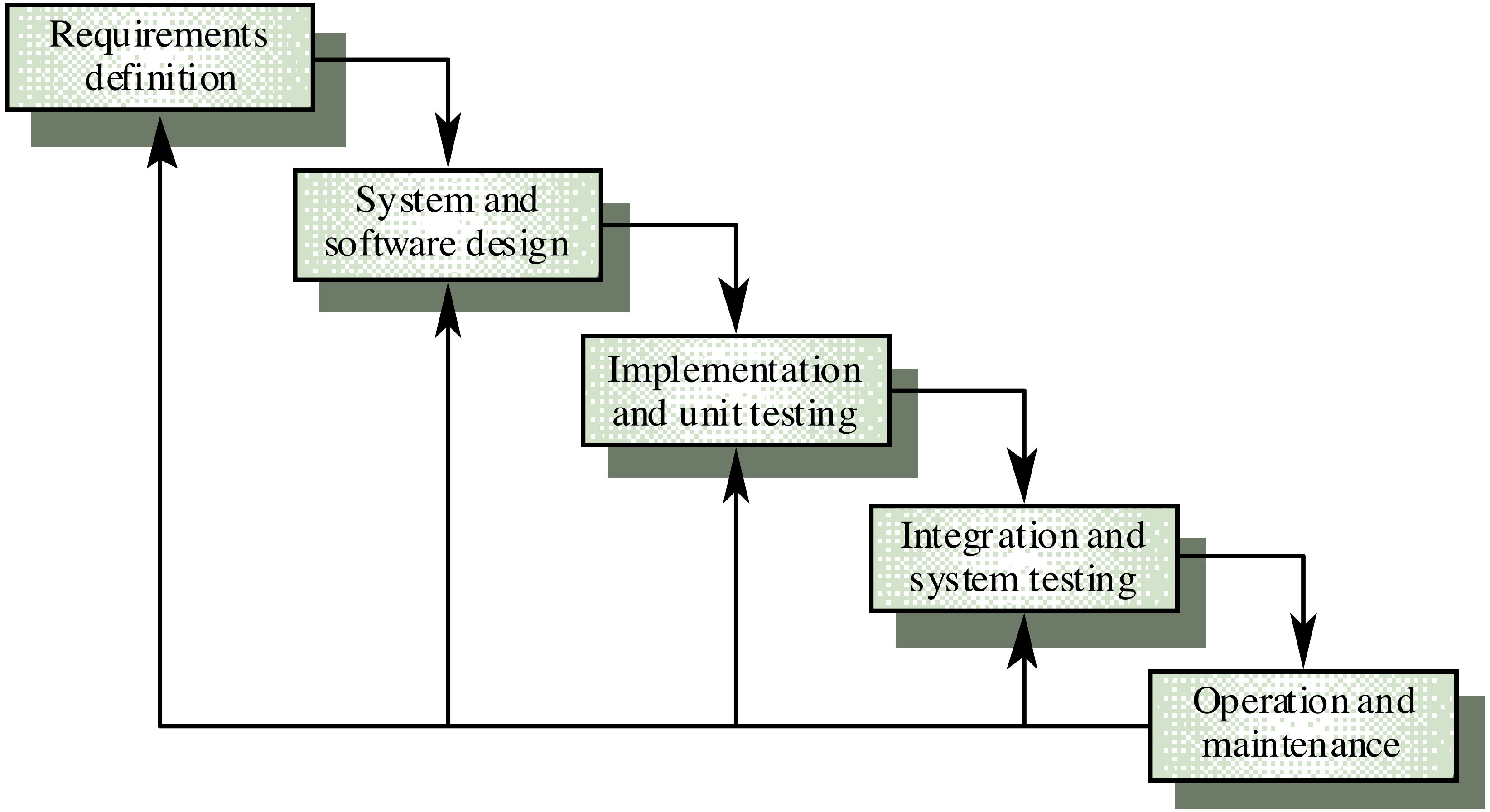
- If prescriptive process models strive for structure and order, are they inappropriate for a software world that thrives on change?
- Yet, if we reject traditional process models (and the order they imply) and replace them with something less structured, do we make it impossible to achieve coordination and coherence in software work?

TRADITIONAL MODELS

- Waterfall
  - a linear framework
- Spiral
  - a combined linear-iterative framework
- Incremental
  - a combined linear-iterative framework or V Model
- Prototyping
  - an iterative framework
- Rapid application development (RAD)
  - an iterative framework



WATERFALL MODEL


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

WATERFALL MODEL

Developers are to follow these phases in order:

Requirements

Software design

Implementation

Testing

Deployment

Maintenance

Each phase is dependent on previous step

Next phase starts only if previous step is finished.

WATERFALL PROCESS CHARACTERISTICS

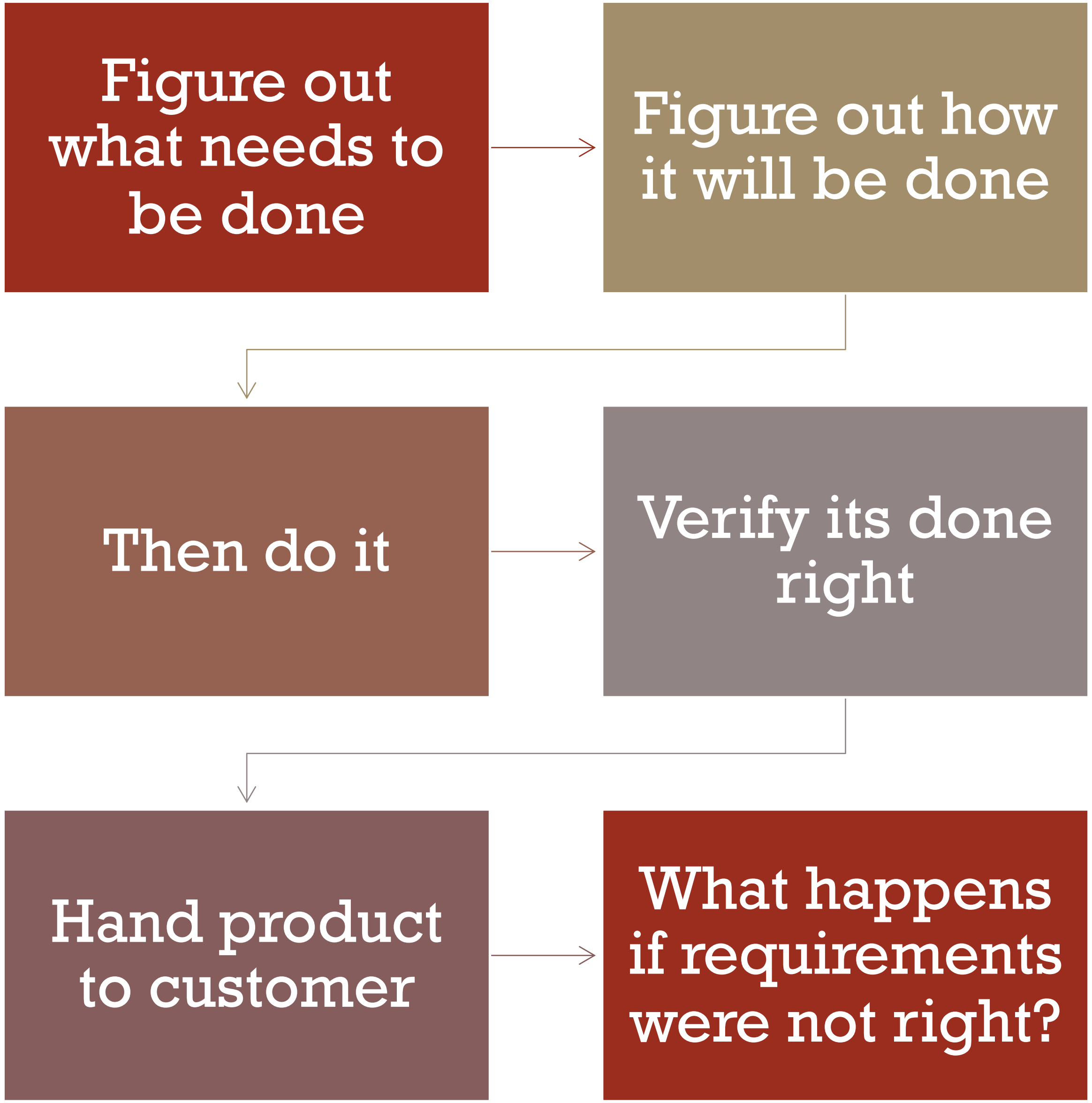


Figure out what needs to be done

Figure out how it will be done

Then do it

Verify its done right

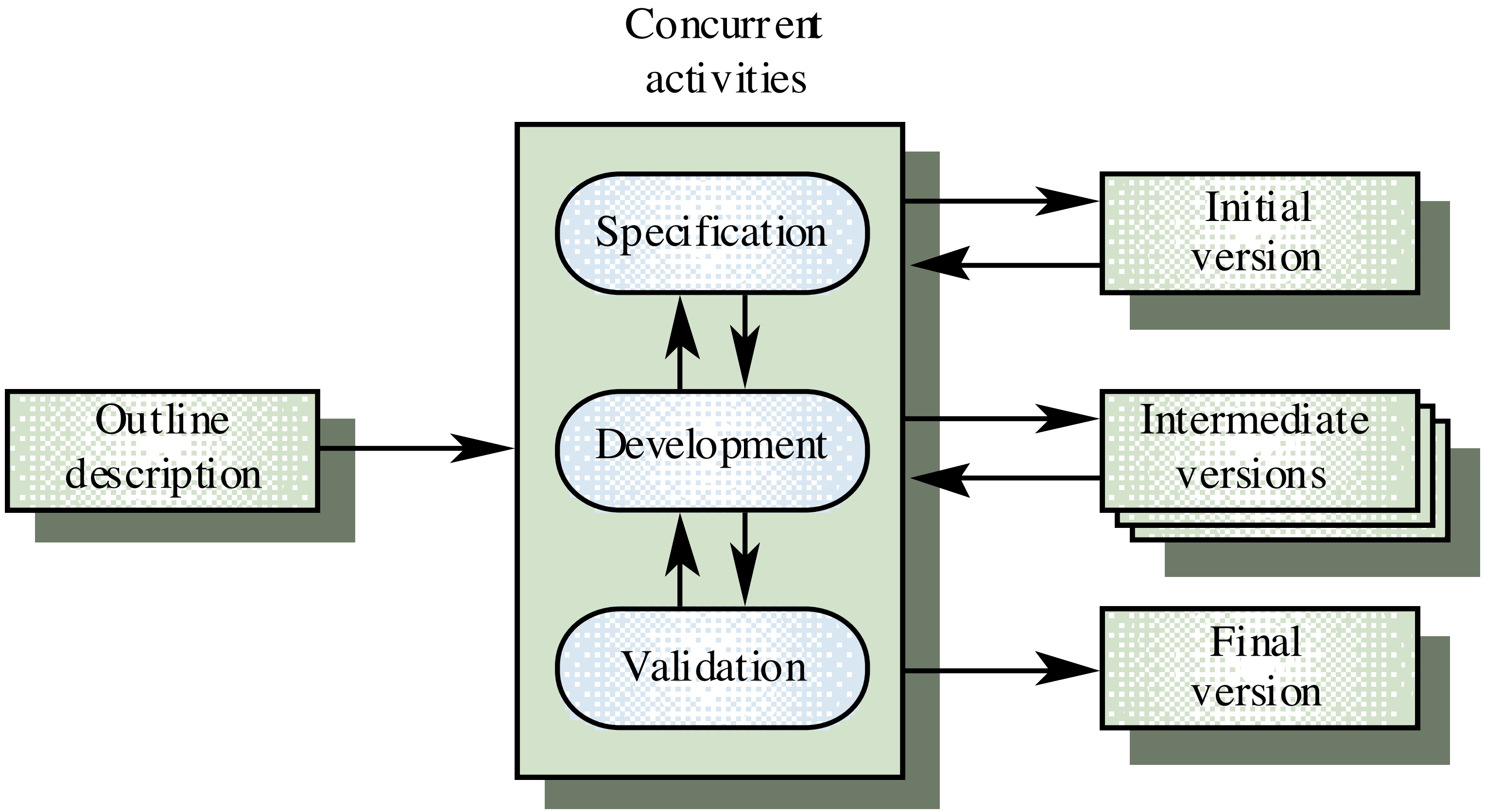
Hand product to customer

What happens if requirements were not right?

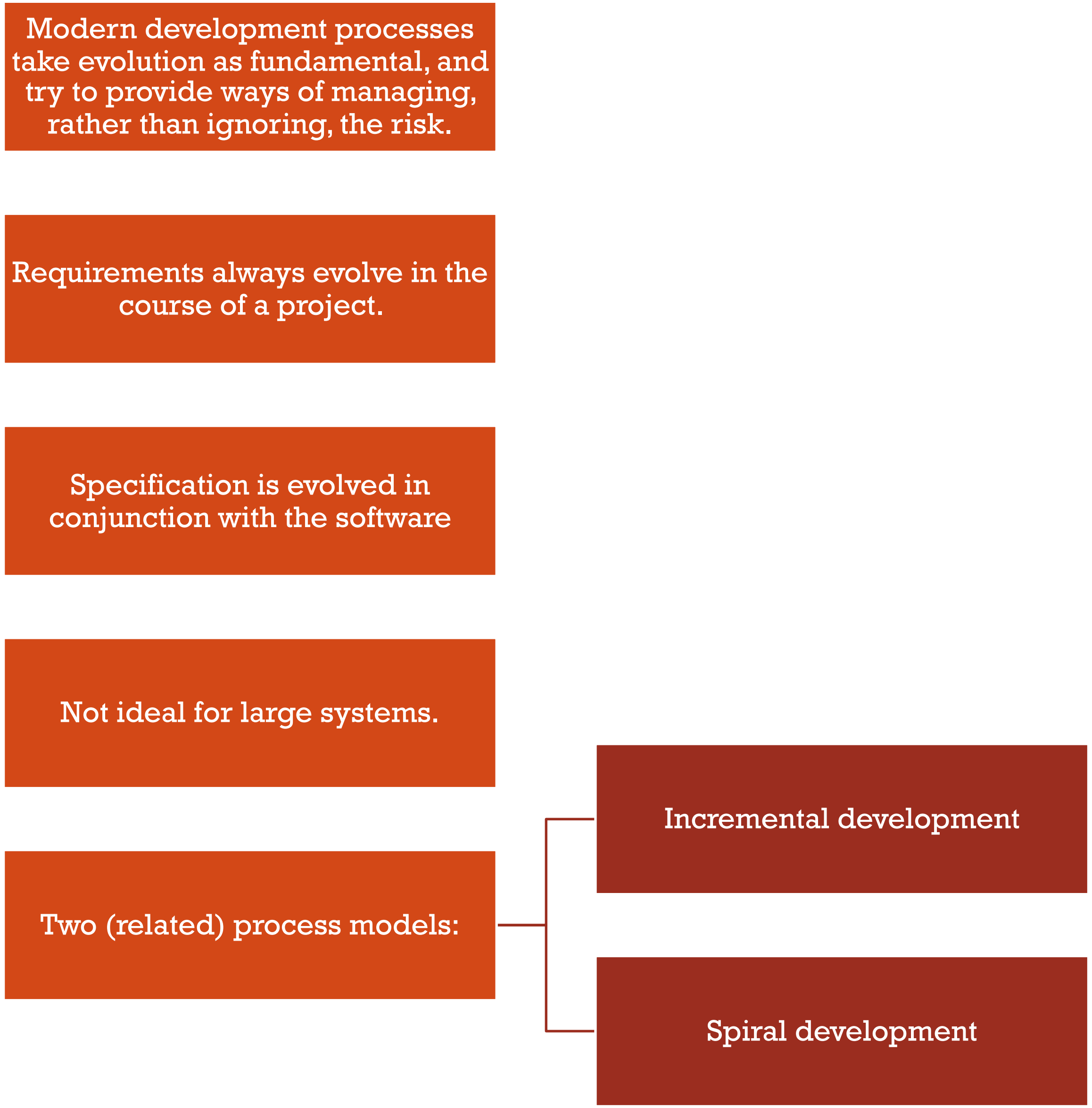
WATERFALL MODEL ISSUES

- Real projects rarely follow the sequential flow that the model proposes.
- At the beginning of most projects requirements are not clear.
- Requirements cannot be changed in the middle.
- The model does not accommodate flexibility very well.
- Development can take very long time and that does not yield a working version of the system until late in the process.

EVOLUTIONARY DEVELOPMENT



EVOLUTIONARY PROCESS CHARACTERISTICS



Modern development processes take evolution as fundamental, and try to provide ways of managing, rather than ignoring, the risk.

Requirements always evolve in the course of a project.

Specification is evolved in conjunction with the software

Not ideal for large systems.

Two (related) process models:

Incremental development

Spiral development

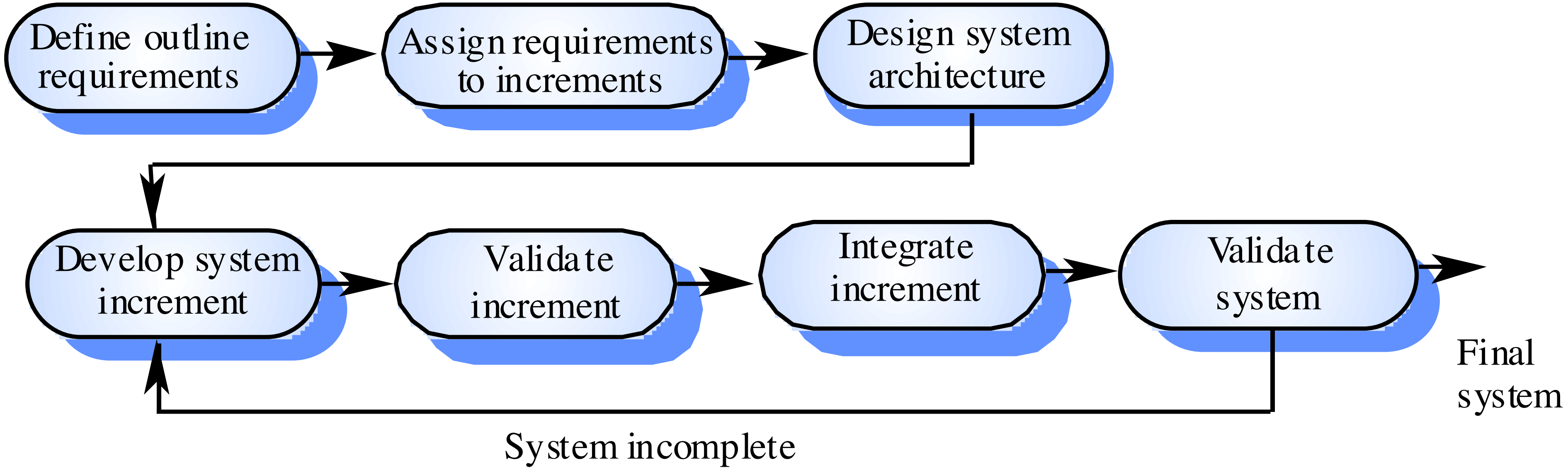
INCREMENTAL DEVELOPMENT

Rather than delivering the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.

Requirements are prioritised and the highest priority requirements are included in early increments.

Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

INCREMENTAL DEVELOPMENT



Define outline requirements

Assign requirements to increments

Design system architecture

Develop system increment

Validate increment

Integrate increment

Validate system

Final system

System incomplete

INCREMENTAL DEVELOPMENT – ADVANTAGES

Customer value can be delivered with each increment so system functionality is available earlier.

Early increments act as a prototype to help elicit requirements for later increments.

Lower risk of overall project failure.

The highest priority system services tend to receive the most testing.

INCREMENTAL DEVELOPMENT – PROBLEMS

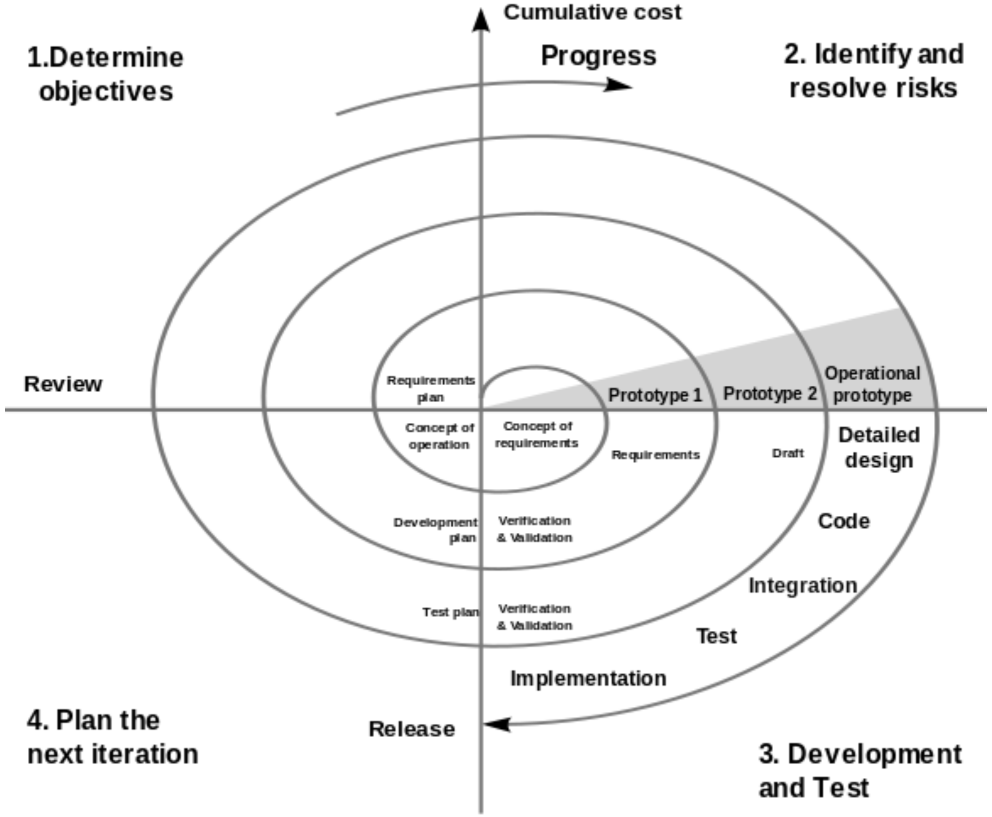
Lack of process visibility.

Systems are often poorly structured.

Not ideal for large systems.

SPIRAL

- The key characteristic of is risk management at regular stages in development cycle
- Combines key aspect of the waterfall model & rapid prototyping
- Good for complex systems.

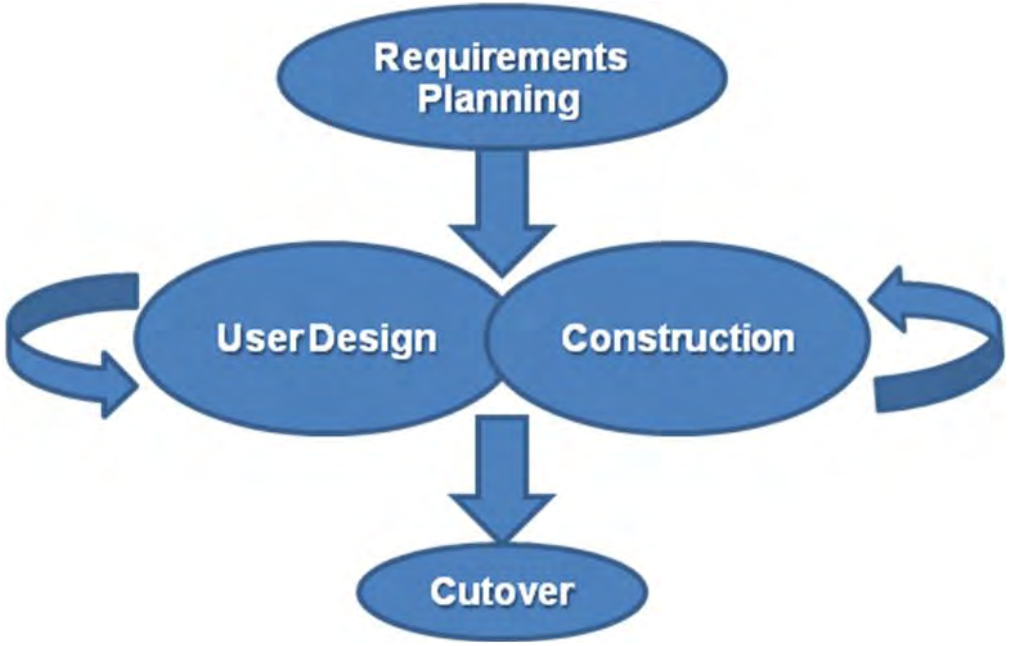


SPIRAL

- Process passing through some number of iterations.
- More emphasis on risk analysis.
- Requires to accept the analysis and act on it.
- Willingness to spend more to fix the issues, which is the reason why this model is often used for large-scale internal software development.
- If the implementation of risk analysis will greatly affect the profits of the project, the spiral model should not be used.

RAPID APPLICATION DEVELOPMENT

- RAD requires minimal planning.
- Faster development.
- Easier to change requirements.
- Iterative & prototyping
- Starts with data models and business process modeling.
- Requirements are verified by prototyping, eventually to refine the data and process models.



AGILE DEVELOPMENT

PROJECT FAILURE – TRIGGER FOR AGILITY

ONE OF THE PRIMARY CAUSES OF PROJECT FAILURE WAS THE EXTENDED PERIOD OF TIME IT TOOK TO DEVELOP A SYSTEM.

COSTS ESCALATED AND REQUIREMENTS CHANGED.

AGILE METHODS INTEND TO DEVELOP SYSTEMS MORE QUICKLY WITH LIMITED TIME SPENT ON ANALYSIS AND DESIGN.

WHAT IS AGILITY?

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed
- Yielding … Rapid, incremental delivery of software

AN AGILE PROCESS

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur

AGILE PROCESS

Agile methods are considered

- Lightweight
- People-based rather than Plan-based

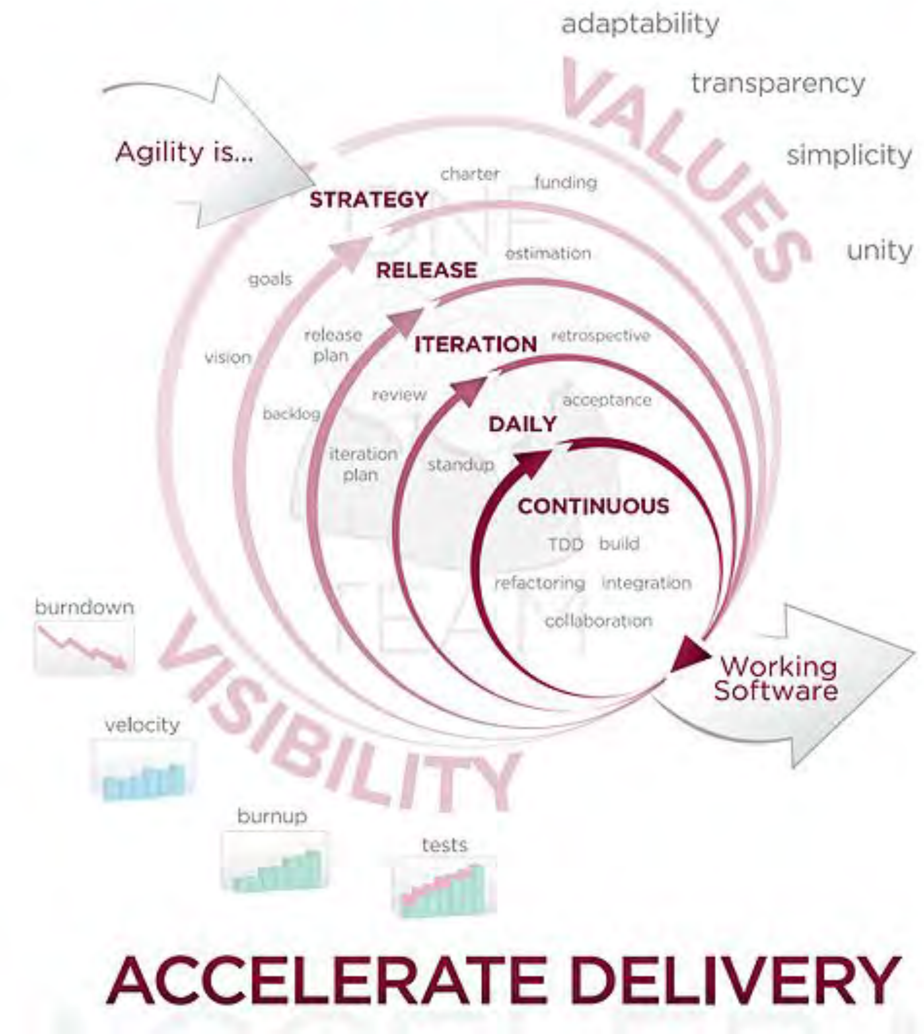
Several agile methods

- Extreme Programming (XP) most popular
- SCRUM
- TDD etc…

Agile Manifesto closest to a definition

- Set of principles
- Developed by Agile Alliance

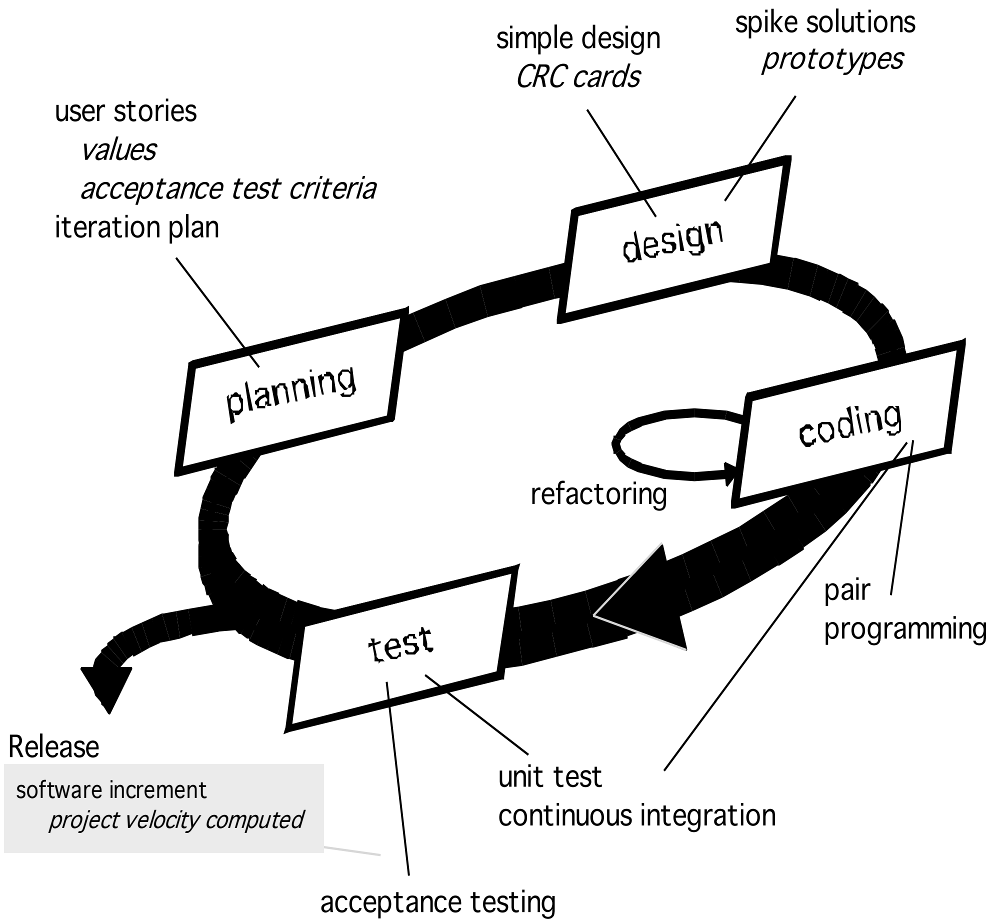
AGILE DEVELOPMENT



EXTREME PROGRAMMING (XP)

- Follows agile process
- The phases are carried out in extremely small (or "continuous")
- First write automated tests as concrete goal for development
- Then coding. Complete only if all tests passed
- Design and architecture emerge out of refactoring
- The incomplete but functional system is deployed or demonstrated
- Move to next part of the system

EXTREME PROGRAMMING (XP)



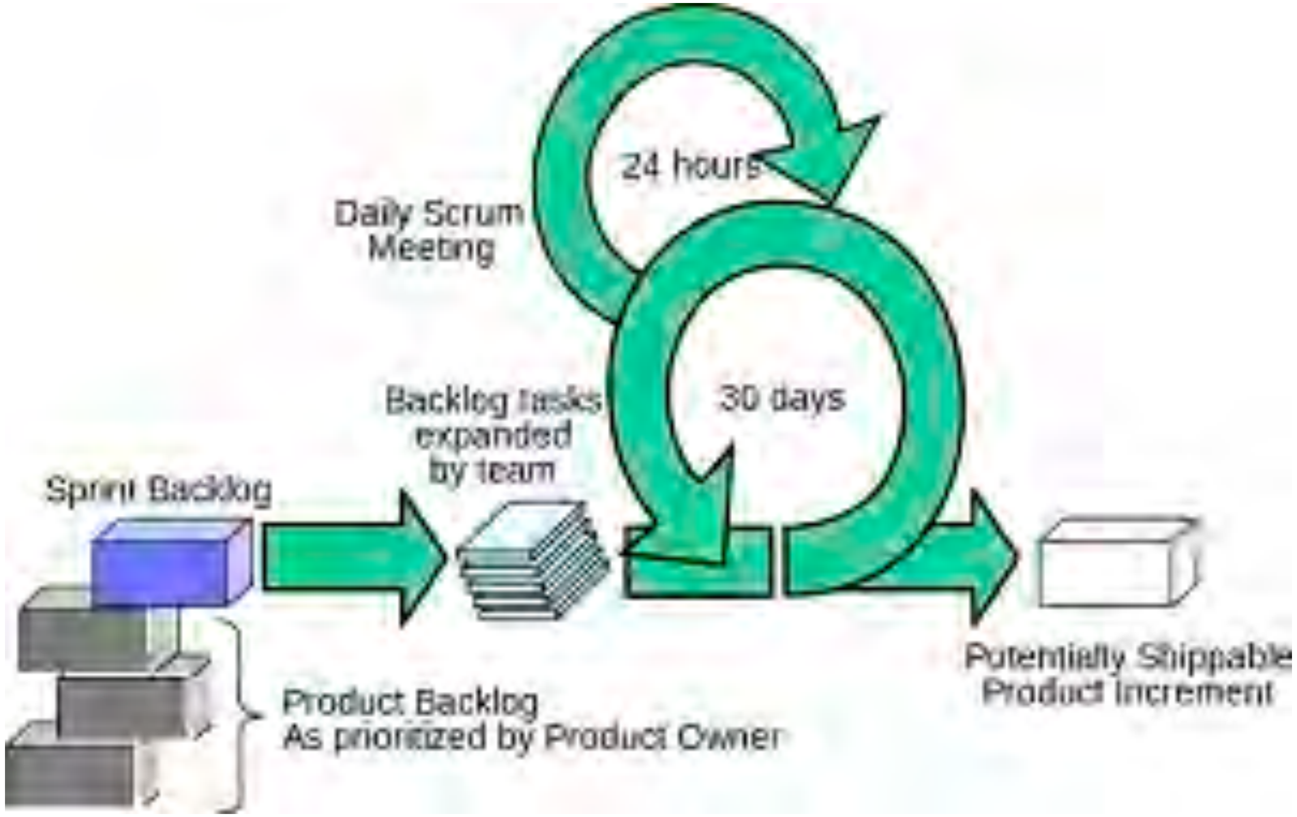
SCRUM

- Scrum is a framework for agile software development
- Enables the creation of self-organizing teams by encouraging co-location of all team members
- Testing and documentation are on-going as the product is constructed
- Work occurs in “sprints” and is derived from a “backlog” of existing requirements
- Meetings are very short and sometimes conducted without chairs
- “demos” are delivered to the customer with the time-box allocated

SCRUM TERMS

Scrum Team	product owner, development team, scrum master
Sprint	Timeboxed iteration of a continuous development cycle
Planning	Work and effort necessary to meet their **sprint** commitment
Product Backlog	List of all things that needs to be done within the project
Sprint Backlog	list of all things that needs to be done within a sprint
Daily Meeting	15-minute meeting to provide status update
Review	**Review** of the team's activities during the **Sprint**
Retrospective	What went well and continue? What can be improved? Actions

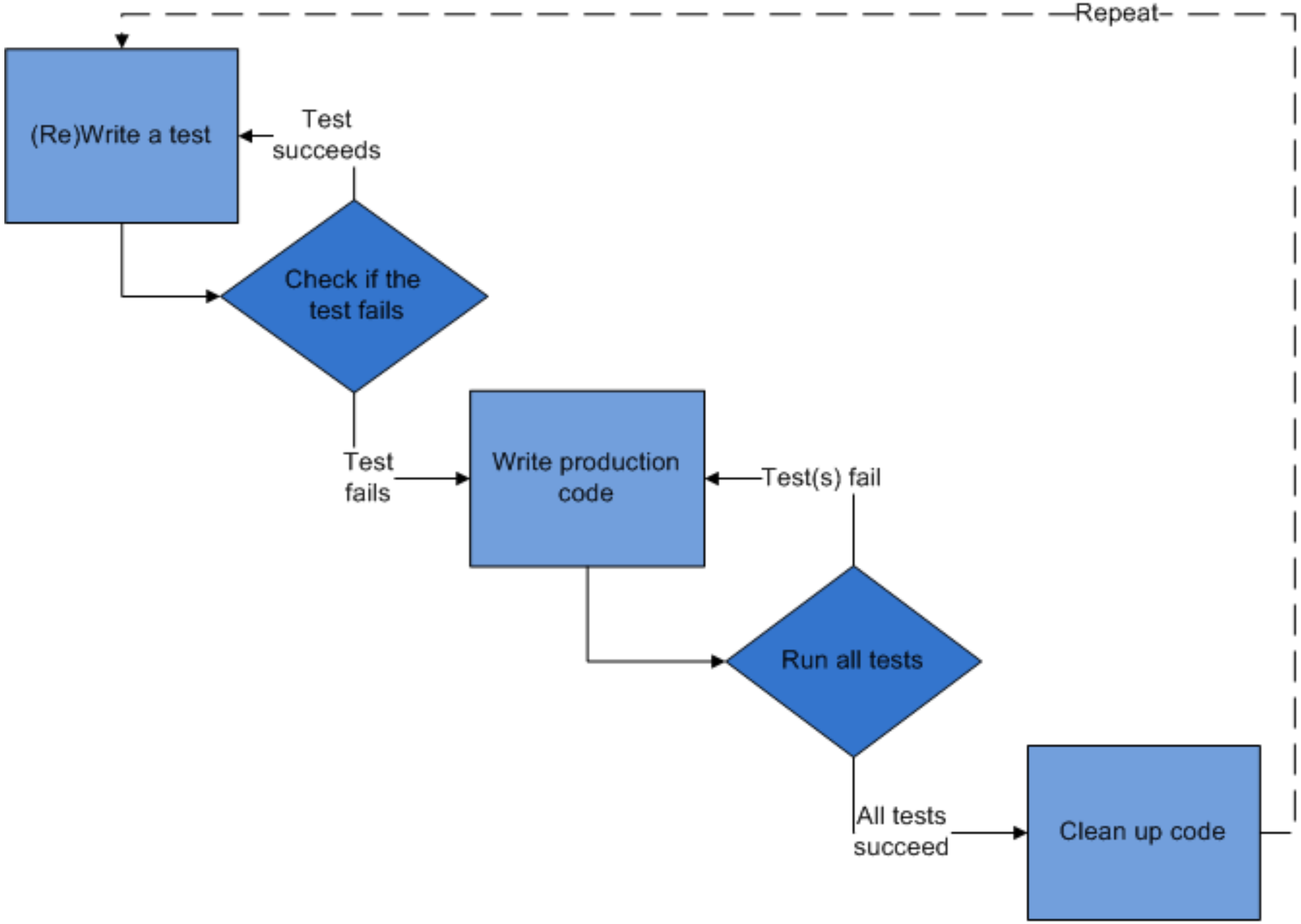
SCRUM – PROCESS FLOW


24 hours
Daily Scrum Meeting
30 days
Backlog tasks expanded by team
Sprint Backlog
Product Backlog
As prioritized by Product Owner
Potentially Shippable Product Increment

TEST DRIVEN DEVELOPMENT

- A process that relies on the repetition of a very short development cycle
- Based on test first programming concept of XP
- First write an (initially failing) automated test case that defines a desired improvement or new function
- Write minimum amount of code to pass the test
- ✓ Finally re-factor the code to acceptable standards

TDD



Repeat
(Re)Write a test
Test succeeds
Check if the test fails
Test fails
Write production code
Test(s) fail
Run all tests
All tests succeed
Clean up code

HUMAN ASPECTS OF SOFTWARE ENGINEERING

TRAITS OF SUCCESSFUL SOFTWARE ENGINEERS

- Sense of individual responsibility
- Acutely aware of the needs of team members and stakeholders
- Brutally honest about design flaws and offers constructive criticism
- Resilient under pressure
- Heightened sense of fairness
- Attention to detail
- Pragmatic

EFFECTIVE SOFTWARE TEAM ATTRIBUTES

- SENSE OF PURPOSE
- SENSE OF INVOLVEMENT
- SENSE OF TRUST
- SENSE OF IMPROVEMENT
- DIVERSITY OF TEAM MEMBER SKILL SETS

AVOID TEAM "TOXICITY"

A frenzied work atmosphere in which team members waste energy and lose focus on the objectives of the work to be performed.

High frustration caused by personal, business, or technological factors that cause friction among team members.

"Fragmented or poorly coordinated procedures" or a poorly defined or improperly chosen process model that becomes a roadblock to accomplishment.

Unclear definition of roles resulting in a lack of accountability and resultant finger-pointing.

"Continuous and repeated exposure to failure" that leads to a loss of confidence and a lowering of morale.

FACTORS AFFECTING TEAM STRUCTURE

- the difficulty of the problem to be solved
- the size of the resultant program(s) in lines of code or function points
- the time that the team will stay together (team lifetime)
- the degree to which the problem can be modularized
- the required quality and reliability of the system to be built
- the rigidity of the delivery date
- the degree of sociability (communication) required for the project

TEAM VALUES

Communication

- close informal verbal communication among team members and stakeholders and continuous feedback

Simplicity

- design for immediate needs nor future needs

Feedback

- derives from the implemented software, the customer, and other team members

Courage

- the discipline to resist pressure to design for unspecified future requirements

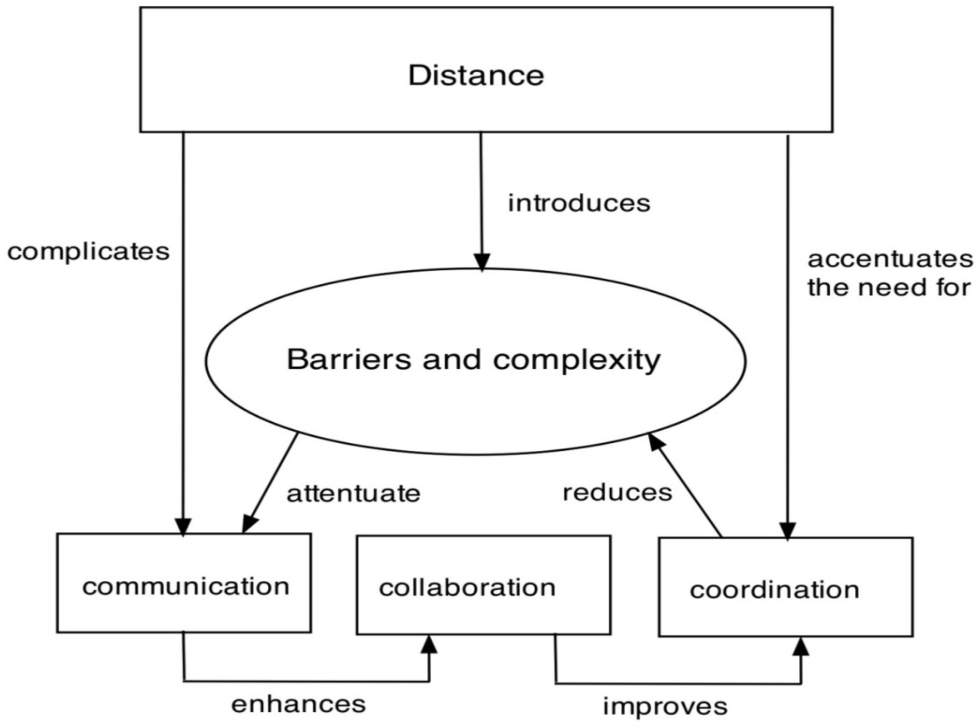
Respect

- among team members and stakeholders

TEAM DECISIONS MAKING COMPLICATIONS

- Problem complexity
- Uncertainty and risk associated with the decision
- Work associated with decision has unintended effect on another project object (law of unintended consequences)
- Different views of the problem lead to different conclusions about the way forward
- Global software teams face additional challenges associated with collaboration, coordination, and coordination difficulties

FACTORS AFFECTING GLOBAL SOFTWARE DEVELOPMENT TEAM



COLLABORATION TOOLS

- Namespace that allows secure, private storage or work products
- Calendar for coordinating project events
- Templates that allow team members to create artifacts that have common look and feel
- Metrics support to allow quantitative assessment of each team member's contributions
- Communication analysis to track messages and isolates patterns that may imply issues to resolve
- Artifact clustering showing work product dependencies

IMPACT OF SOCIAL MEDIA

Blogs – can be used share information with team members and customers

Microblogs (e.g. Twitter) – allow posting of real-time messages to individuals following the poster

Targeted on-line forums – allow participants to post questions or opinions and collect answers

Social networking sites (e.g. Facebook, LinkedIn) – allows connections among software developers for the purpose of sharing information

Social book marking (e.g. Delicious, Stumble, CiteULike) – allow developers to keep track of and share web-based resources

SOFTWARE ENGINEERING USING THE CLOUD

Benefits

- Provides access to all software engineering work products
- Removes device dependencies and available every where
- Provides avenues for distributing and testing software
- Allows software engineering information developed by one member to be available to all team members

Concerns

- Reliability and security risks
- Potential for interoperability problems
- Usability and performance

REFERENCE

- Roger Pressman, Software Engineering: A Practitioner's Approach, 8th edition, McGraw Hill, ISBN 0078022126