Software Engineering

Prepared By
Sachin D. Shelke

Course Objectives

To learn the principles of Software Engineering.

To learn and understand methods of capturing, specifying, visualizing and analyzing software requirements.

To know design principles to software project development.

To learn basics of IT project management.

To understand software quality attributes and testing principles.

To introduce formal methods and recent trends in Software Engineering.

Course Outcomes

Classify various software application domains.

Analyze software requirements by using various modeling techniques.

Translate the requirement models into design models.

Apply planning and estimation to any project.

Use quality attributes and testing principles in software development life cycle.

Discuss recent trends in Software engineering by using CASE and agile tools.

Syllabus

Unit 1 : Introduction to Software Engineering Process

Unit 2 : Requirement Engineering and Analysis

Unit 3 : Design Engineering

Unit 4 : Project Planning, Management and Estimation

Unit 5 : Software Quality and Testing

Unit 6 : Formal Methods, Recent Trends in Software Engineering

Syllabus

Unit 1 : Introduction to Software Engineering Process

Software Engineering Fundamentals: Nature of Software, Software Engineering Practice, Software Process, Software Myths.

Process Models : A Generic Process Model, Linear Sequential Development Model, Iterative Development Model, The incremental Development Model

Agile software development: Agile manifesto, agility principles, Agile methods, myth of planned development, Introduction to Extreme programming and Scrum.

Agile Practices: test driven development, pair programming, continuous integration in DevOps , Refactoring

Case Study An information system – Library Management system

Syllabus

Unit 2 : Requirements Engineering & Analysis

Requirements Engineering: User and system requirements, Functional and non-functional requirements, requirements engineering (elicitation, specification, validation, negotiation) prioritizing requirements (Kano diagram), requirement traceability matrix(RTM)

Software Requirements Specification (SRS): software requirements Specification document, Software Requirements Specification (SRS): software requirements Specification document

Requirements Analysis: Analysis Model, data modeling, scenario based modeling, class based modeling, Flow oriented modeling, behavioral modeling-Introduction to UML Diagrams

Case Study : Library Management system

Syllabus

Unit 3 : Design Engineering

Design Engineering : Design Process & quality, Design Concepts, design Model, Pattern-based Software Design. Architectural Design :Design Decisions, Views, Patterns, Application Architectures,

Component level Design: component, Designing class based components, conducting component-level design, User Interface Design: The golden rules, Interface Design steps& Analysis, Design Evaluation

Case Study : Library Management system

Syllabus

Unit 4 : Project Planning, Management And Estimation

Project Planning: Project initiation, Planning Scope Management, Creating the Work Breakdown Structure, scheduling: Importance of Project Schedules, Developing the Schedule using Gantt Charts, PERT/ CPM

Project Management: The Management Spectrum, People, Product, Process, Project, The W5HH Principle, Metrics in the Process and Project Domains, Software Measurement: size & function oriented metrics(FP & LOC), Metrics for Project

Project Estimation: Software Project Estimation, Decomposition Techniques, Cost Estimation Tools and Techniques, Typical Problems with IT Cost Estimates.

Case Study : Project Management tool like OpenProj or MS Project

Syllabus

Unit 5 : Software Quality And Testing

Quality Concepts: Quality, software quality, Quality Metrics, software quality dilemma, achieving software quality

Software Testing: Introduction to Software Testing, Principles of Testing, Test plan, Test case, Types of Testing, Verification & Validation, Testing strategies, Defect Management, Defect Life Cycle, Bug Reporting, debugging.

Project Estimation: Software Project Estimation, Decomposition Techniques, Cost Estimation Tools and Techniques, Typical Problems with IT Cost Estimates.

Case Study : Software testing tool like selenium

Syllabus

Unit 6 : Formal Methods Recent Trends In Software Engineering

Recent Trends in SE : SCM, Risk Management, Technology evolution, process trends, collaborative development, software reuse, test-driven development, global software development challenges, CASE – taxonomy, tool-kits, workbenches, environments, components of CASE, categories (upper, lower and integrated CASE tools), Introduction to agile tools Jira, Kanban

Case Study : CASE software/ HP Quality Center (QC) / Jira

Text Books

1. *Roger Pressman, "Software Engineering: A Practitioner's Approach", McGraw Hill, ISBN 0-07- 337597-7*

2. *Rajib Mall, "Fundamentals of Software Engineering", Prentice Hall India, ISBN-13:9788-1203- 4898-1*

3. **https://nptel.ac.in/courses/106/105/106105182/**

4. **http://www2.cs.uh.edu/~rsingh/documents/software_design/TDD.pdf**

What is Software?

Software encompasses: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately store and manipulate information and (3) documentation that describes the operation and use of the programs.

Software Products

- Generic products
  - Stand-alone systems that are marketed and sold to **any customer** who wishes to buy them.
  - Examples – PC software such as editing, graphics programs, project management tools; software for specific markets such as appointments systems for dentists.
- Customized products
  - Software that is commissioned by **a specific customer** to meet their own needs.
  - Examples – embedded control systems, traffic monitoring systems.

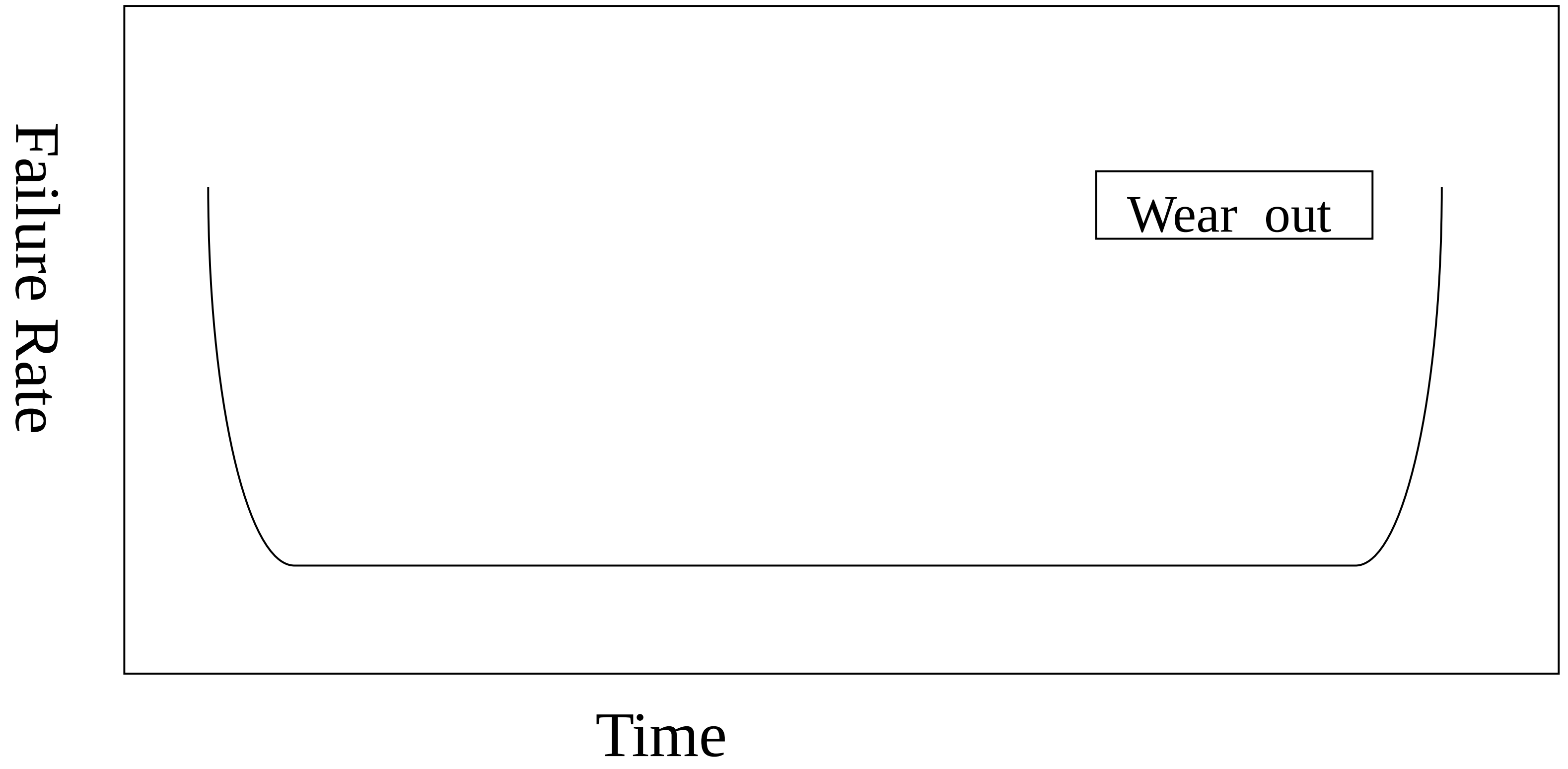
Features of Software?

- Its characteristics that make it different from other things human being build.

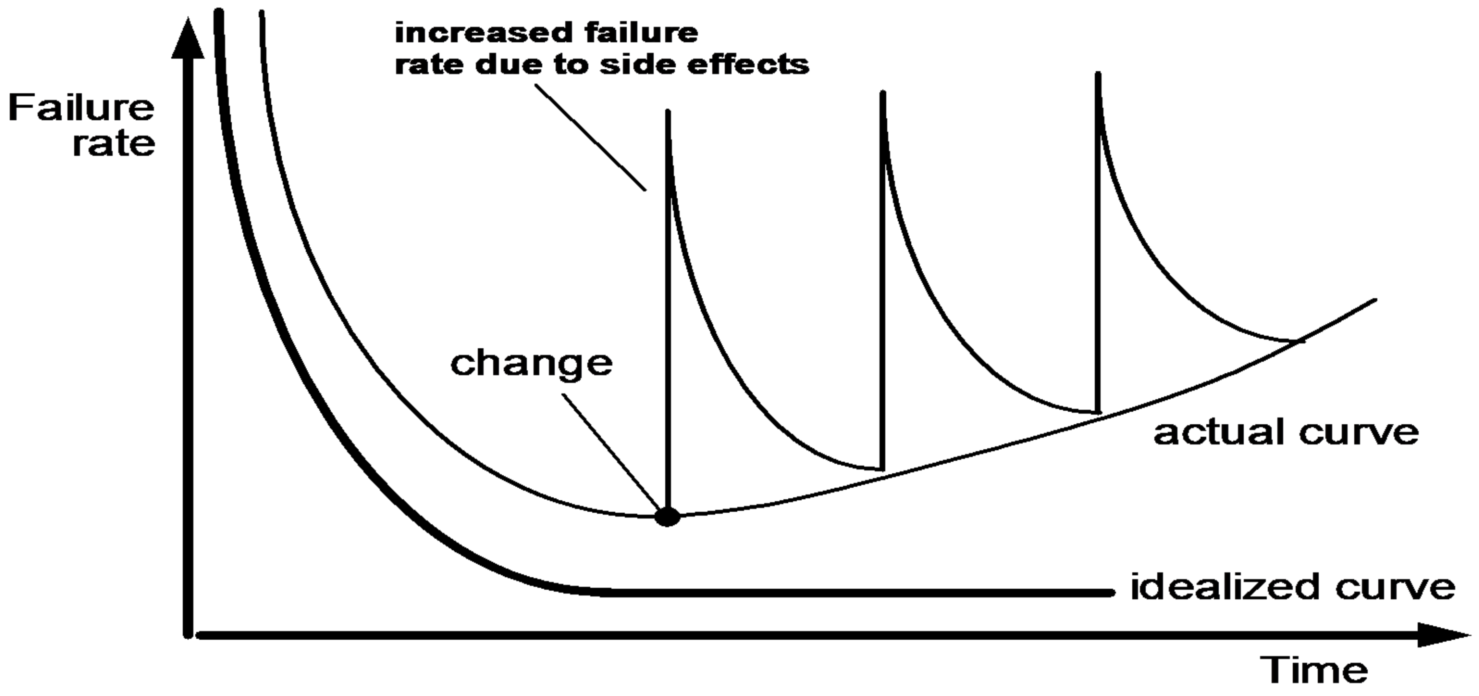
Features of such logical system:

- Software is developed or engineered, it is not manufactured in the classical sense which has quality problem.
- Software doesn't "wear out." but it deteriorates (due to change). Hardware has bathtub curve of failure.
- Although the industry is moving toward component-based construction , most software continues to be custom-built.

Failure (“Bathtub”)Curve for Hardware



Wear vs. Deterioration



Software Applications

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- Web-Apps
- AI

Software Applications

- System software: such as compilers, editors, file management utilities
- Application software: stand-alone programs for specific needs.
- Engineering/scientific software: Characterized by "number crunching" algorithms such as automotive stress analysis, molecular biology, orbital dynamics etc
- Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
- Product-line software focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
- WebApps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.
- AI software uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition game playing

Software—New Categories

- Open world computing — distributed computing due to wireless networking. How to allow mobile devices, personal computer, enterprise system to **communicate across vast network**.
- Netsourcing — the Web as a computing engine. How to architect simple and sophisticated applications to target end-users worldwide.
- Open source — ”free” source code open to the computing community
- Cognitive machines

Software Engineering Definition

The Seminal definition:

[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

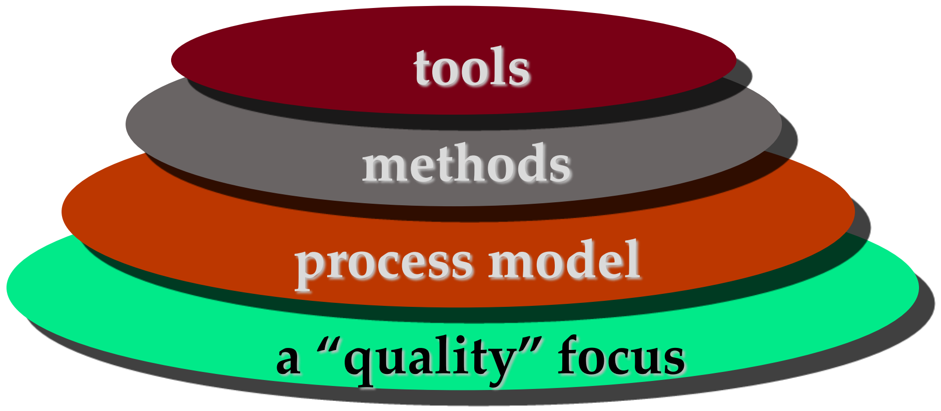
The IEEE definition:

Software Engineering: The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

Importance of Software Engineering

- More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

Software Engineering: A Layered Technology

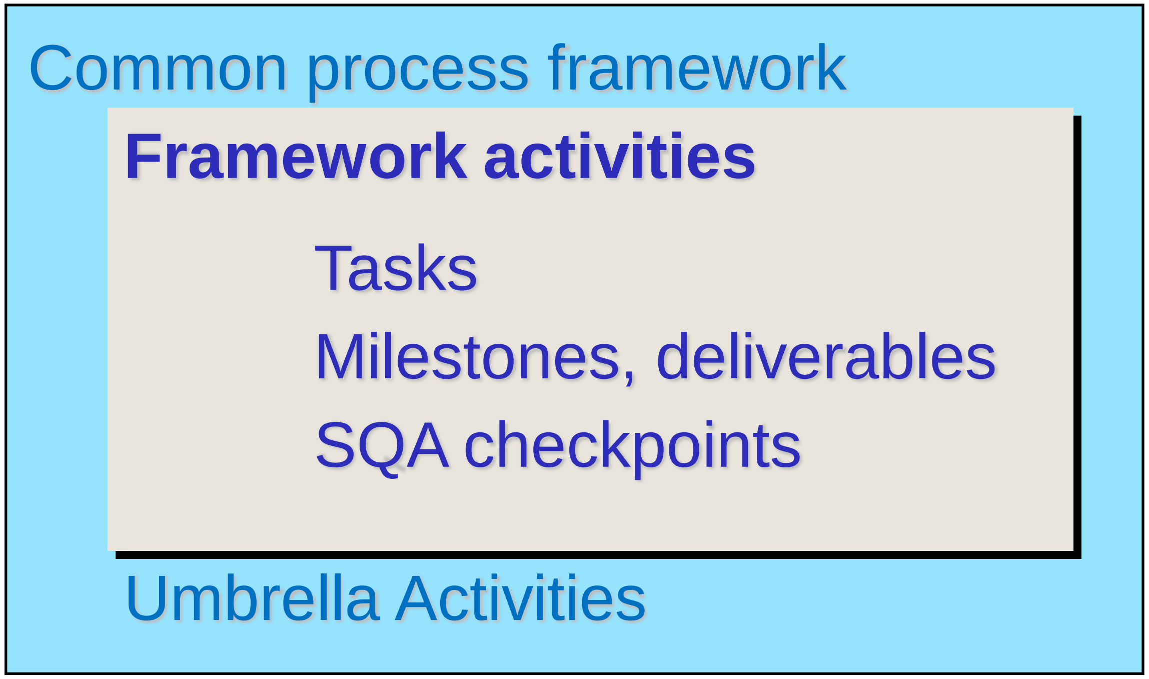


- Any engineering approach must rest on organizational commitment to **quality** which fosters a continuous process improvement culture.

Software Engineering: A Layered Technology

- **Process** layer as the foundation defines a framework with activities for effective delivery of software engineering technology. Establish the context where products (model, data, report, and forms) are produced, milestone are established, quality is ensured and change is managed.
- **Method** provides technical how-to's for building software. It encompasses many tasks including communication, requirement analysis, design modeling, program construction, testing and support.
- **Tools** provide automated or semi-automated support for the process and methods.

A Common Process Framework



Framework Activities

- Communication
- Planning
- Modeling
- Construction
- Deployment

Umbrella Activities

- Software project tracking and control: assess progress against the plan and take actions to maintain the schedule.
- Risk management: assesses risks that may affect the outcome and quality.
- Software quality assurance: defines and conduct activities to ensure quality.
- Technical reviews: assesses work products to uncover and remove errors before going to the next activity.
- Measurement: define and collects process, project, and product measures to ensure stakeholder's needs are met.
- Software configuration management: manage the effects of change throughout the software process.
- Reusability management: defines criteria for work product reuse and establishes mechanism to achieve reusable components.
- Work product preparation and production: create work products such as models, documents, logs, forms and lists.

Software Myths [Management Myths]

- Myth 1: We already have a book that's full of standards and procedures for building the software.
- Myth 2: If we get behind the schedule, we can add more programmers and catch up.
- Myth 3: If I decide to outsource the software project to a third party, I can just relax and let that firm built it.

Software Myths [Practitioner’s Myths]

- Myth 1: Once we write the program and get it to work, our job is done.
- Myth 2: Until I get the program running, I have no way of assessing its quality.
- Myth 3: software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

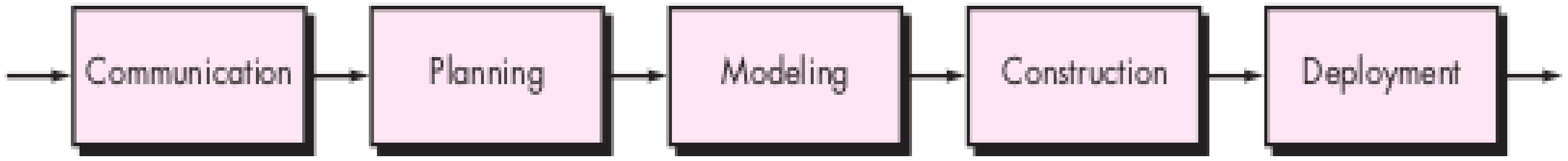
Software Myths [Customer Myths]

- Myth 1: A general statement of objectives is sufficient to begin writing programs- We can fill in the details later.

- Myth 2: Software requirements continually change, but the change can be easily accommodated because software is flexible.

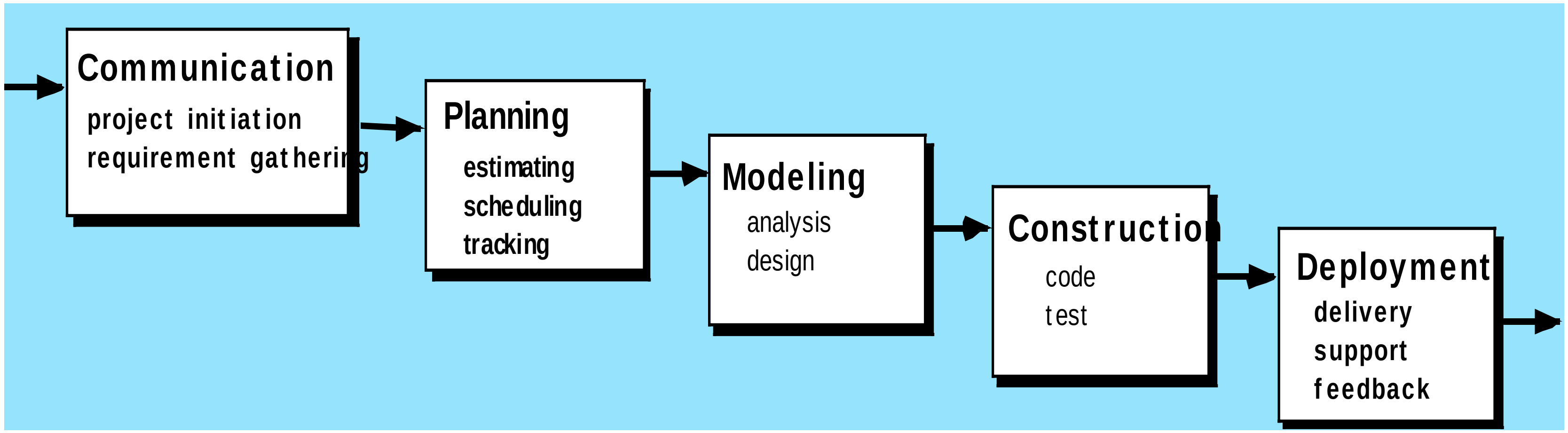
Process Models

A Generic Process Model



(a) Linear process flow

The Waterfall Model



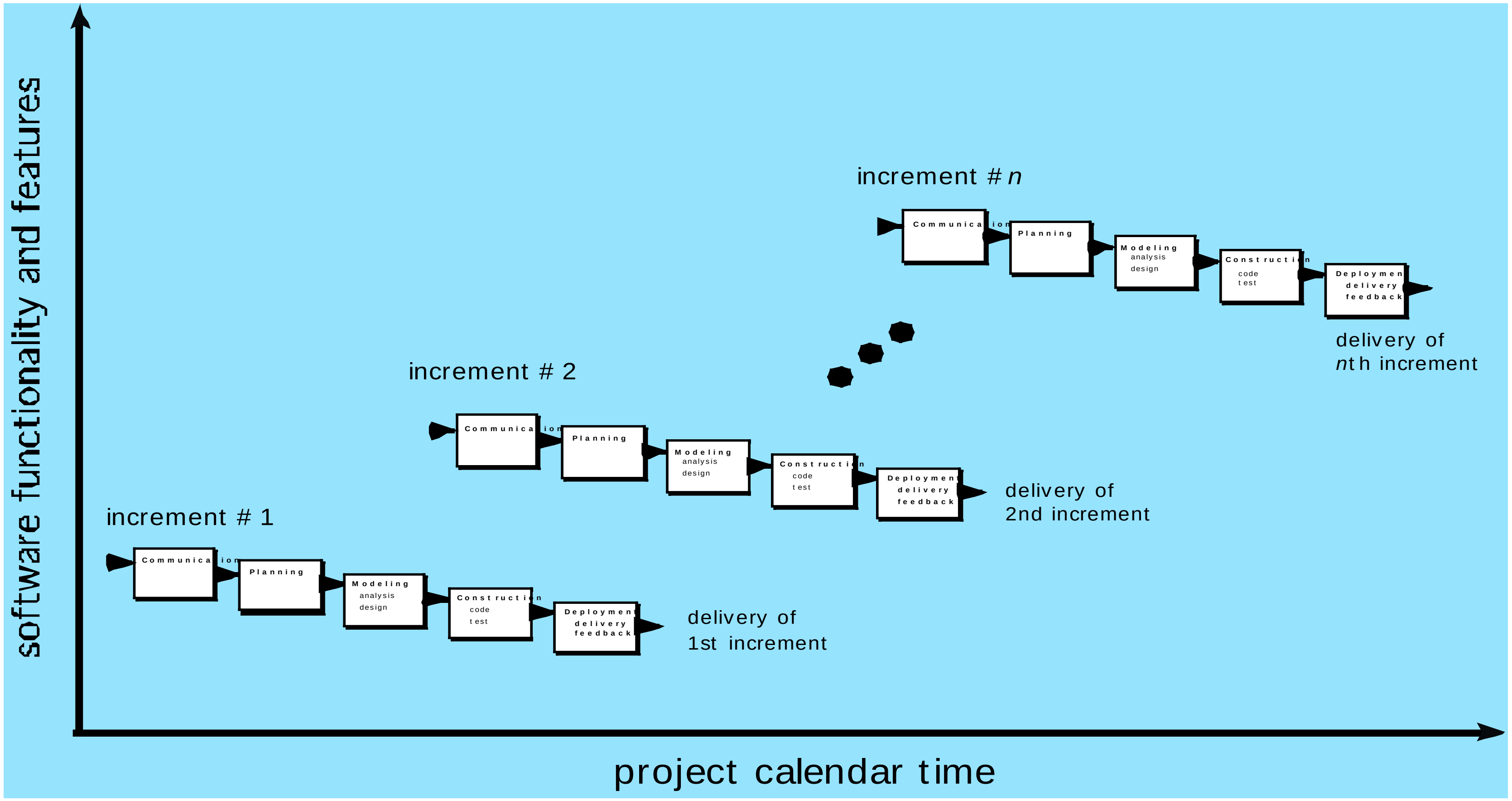
The Waterfall Model

The waterfall process model is oldest paradigm for the software engineering.

Disadvantages:

- Real projects rarely follow the sequential flow.
- It is often difficult for customer to state all requirements explicitly.
- The customer must have patience. A working version of program(s) will not be available until late in the project time span.

The Incremental Model



software functionality and features
increment # 1
Communication
Planning
Modeling analysis design
Construction code test
Deployment delivery feedback
delivery of 1st increment
increment # 2
delivery of 2nd increment
increment # *n*
delivery of *n*th increment
project calendar time

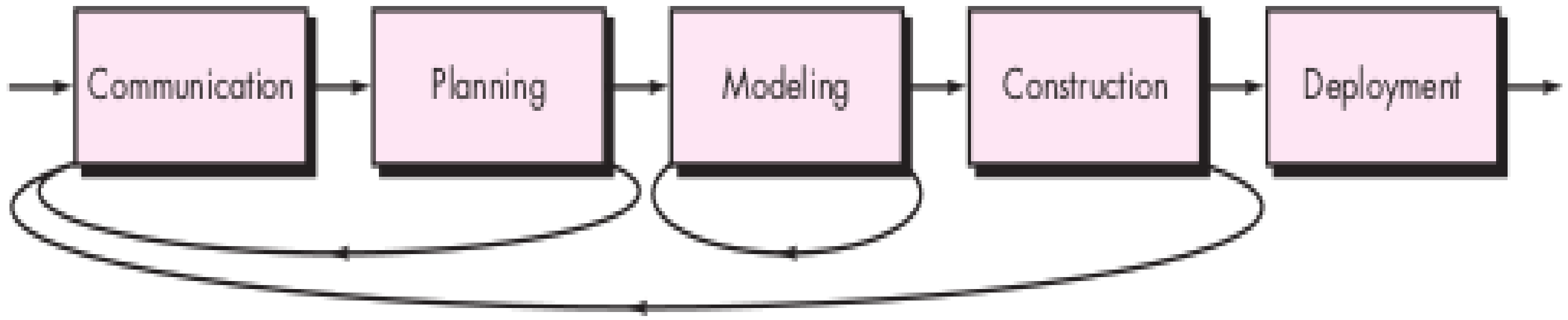
The Incremental Model

Advantages:

- Useful when staffing is unavailable. Early increments can be implemented with fewer people. More staff may be added later in the process.

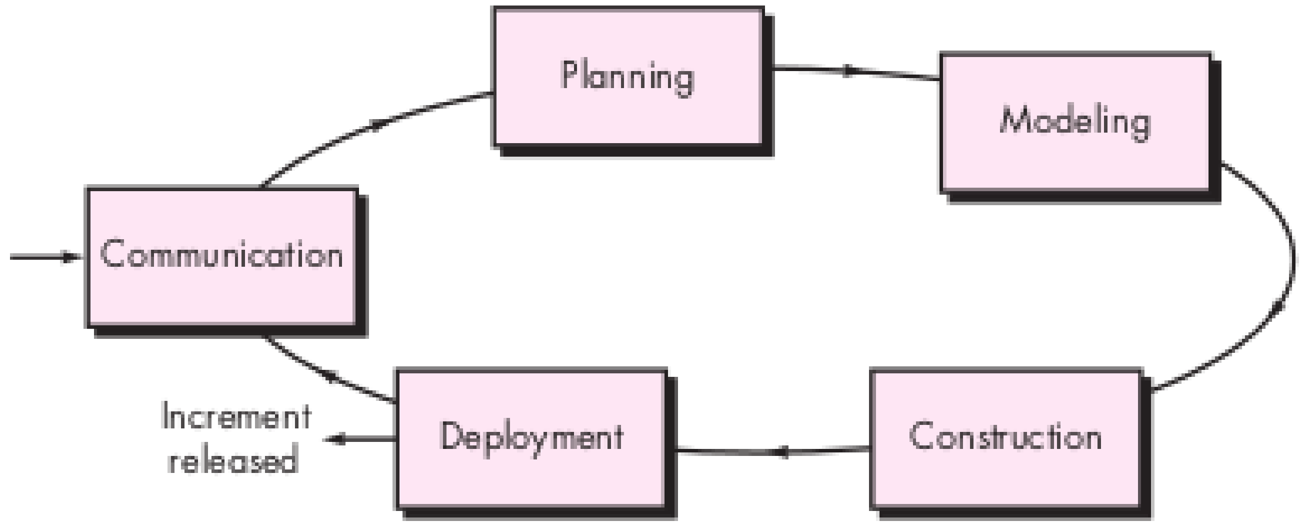
- Increments can be planned to manage technical risks. (eg. availability date of new hardware is uncertain)

The Iterative Process Flow



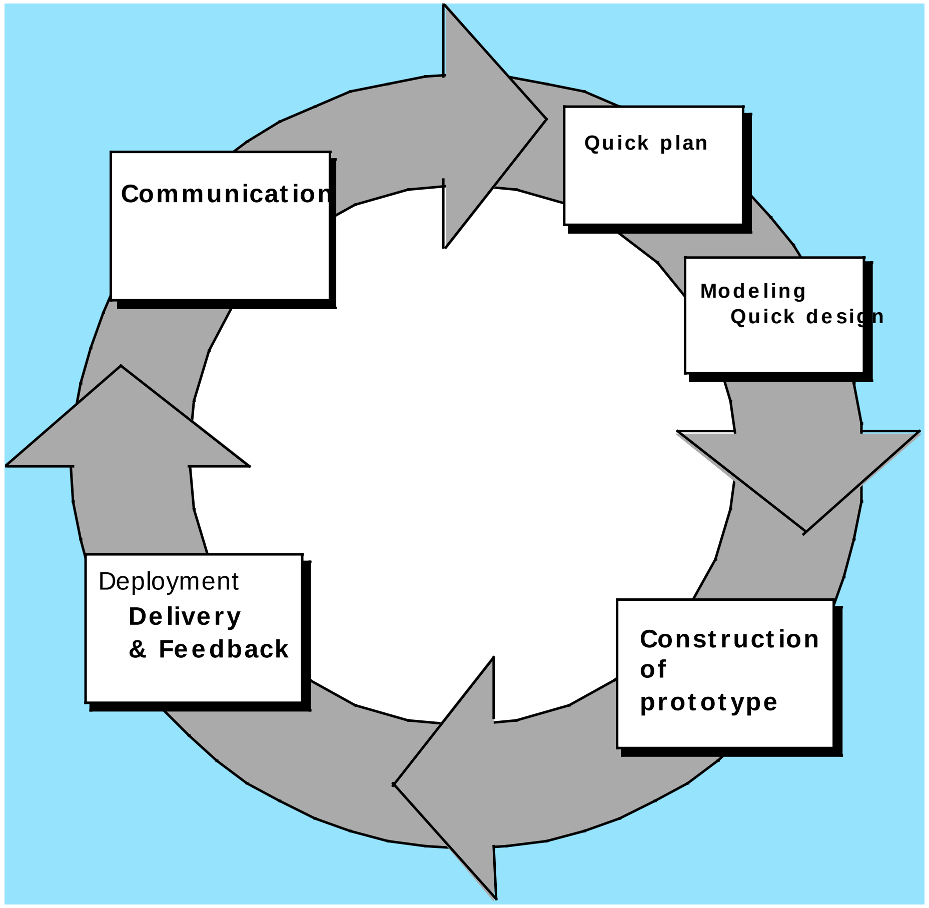
(b) Iterative process flow

The Evolutionary Process Flow



(c) Evolutionary process flow

Evolutionary Models: Prototyping



Evolutionary Models: Prototyping

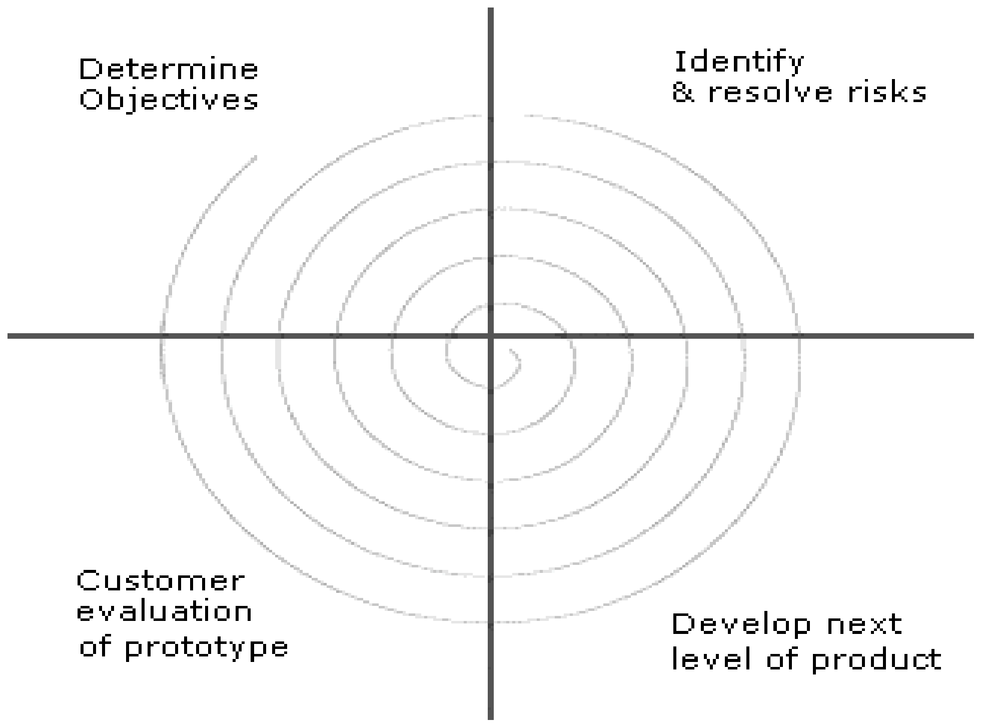
Advantages:

- The prototyping serves as a mechanism for identifying software requirements.
- Prototype can serve as " the first system"

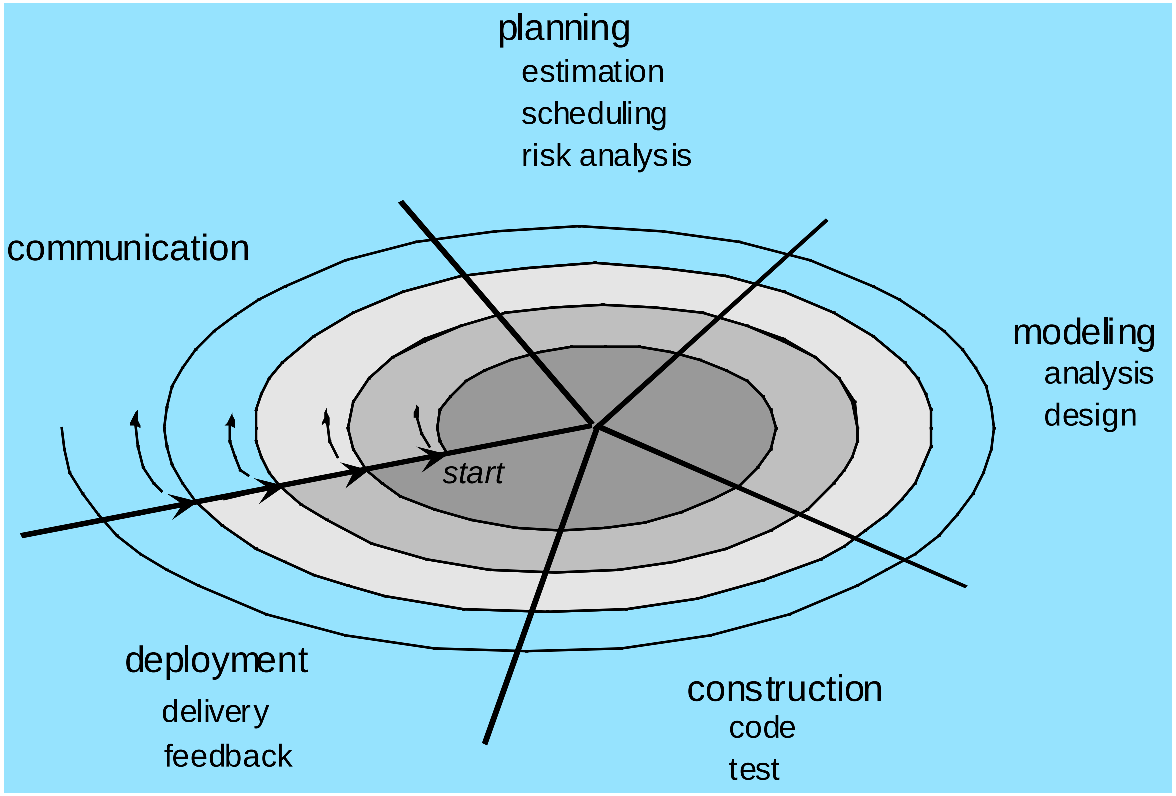
Disadvantages:

- Prototype is built quickly without considering overall software quality , maintainability.
- Inappropriate tools/ programming languages may be used.

Evolutionary Models: The Spiral


Determine Objectives
Identify & resolve risks
Customer evaluation of prototype
Develop next level of product

Evolutionary Models: The Spiral



Evolutionary Models: The Spiral

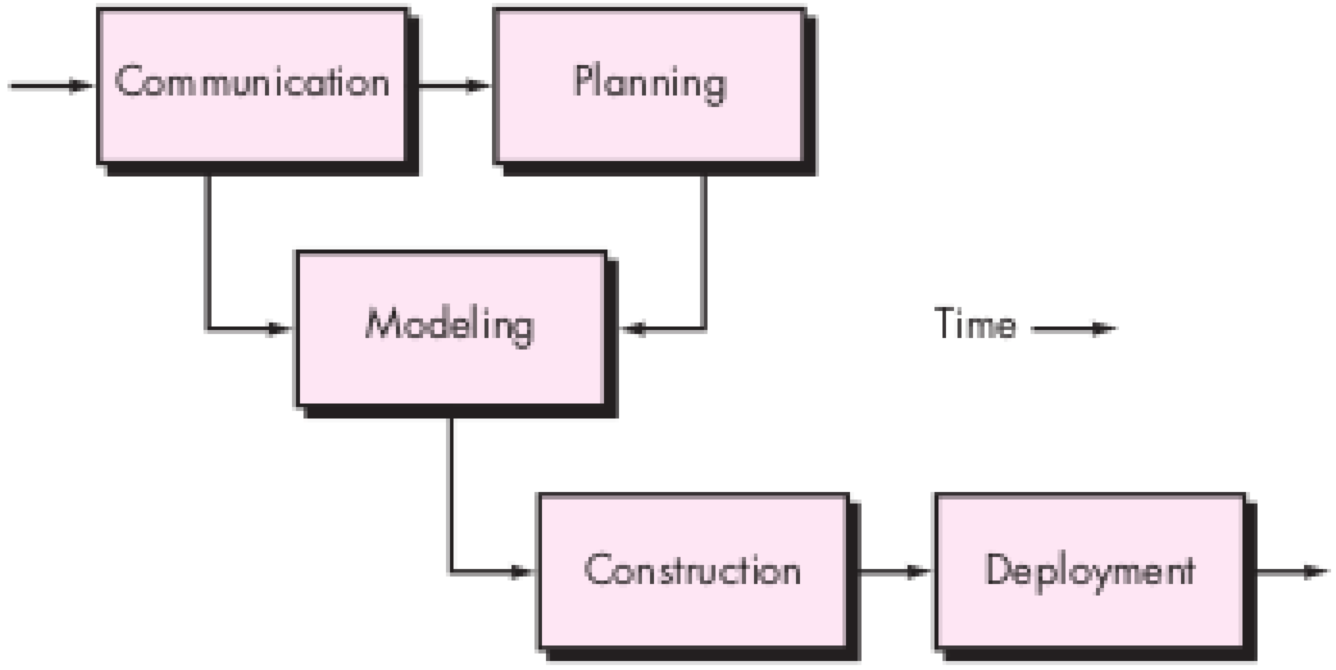
Advantages:

- Realistic approach to the development of large scale systems.
- Uses prototyping as risk reduction mechanism.

Disadvantages:

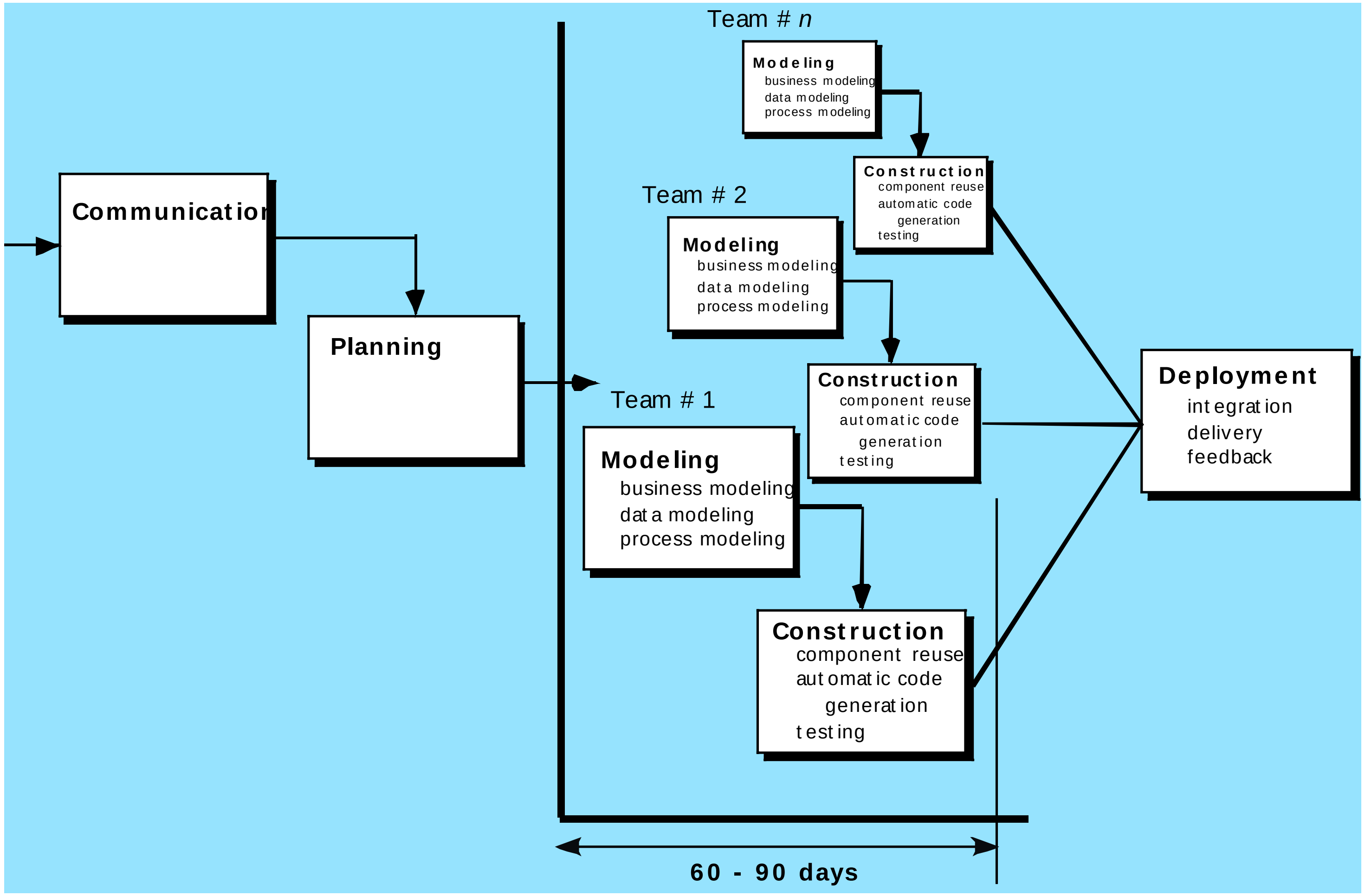
- It may be difficult to convince customers that the evolutionary approach is controllable.

The Parallel Process Flow



(d) Parallel process flow

The RAD Model



The RAD Model

Advantages:

- Quick development of software products.

Disadvantages:

- High skilled resources required.
- Proper modularization of project required.
- Difficult to manage
- Not appropriate when technical risks are high.

Agile Software Development

Agile Software Development

- Classical methods of software development
  - huge effort during the planning phase
  - poor requirements conversion in a rapid changing environment
  - treatment of staff as a factor of production

- New methods:

  Agile Software Development Methodology

The Manifesto for Agile Software Development

- "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
- That is, while there is value in the items on the right, we value the items on the left more."

http://www.agilemanifesto.org

What is “Agility”?

- Effective (rapid and adaptive) response to change
- Effective communication in structure and attitudes among all team members, technological and business people, software engineers and managers。
- Drawing the customer into the team. Eliminate “us and them” attitude.
- Planning in an uncertain world has its limits and plan must be flexible.
- Organizing a team so that it is in control of the work performed

Agility Principles - I

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

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.

Agility Principles - II

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

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.

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.

Agile Methods

- Scrum
- Extreme Programming
- Adaptive Software Development (ASD)
- Dynamic System Development Method (DSDM)

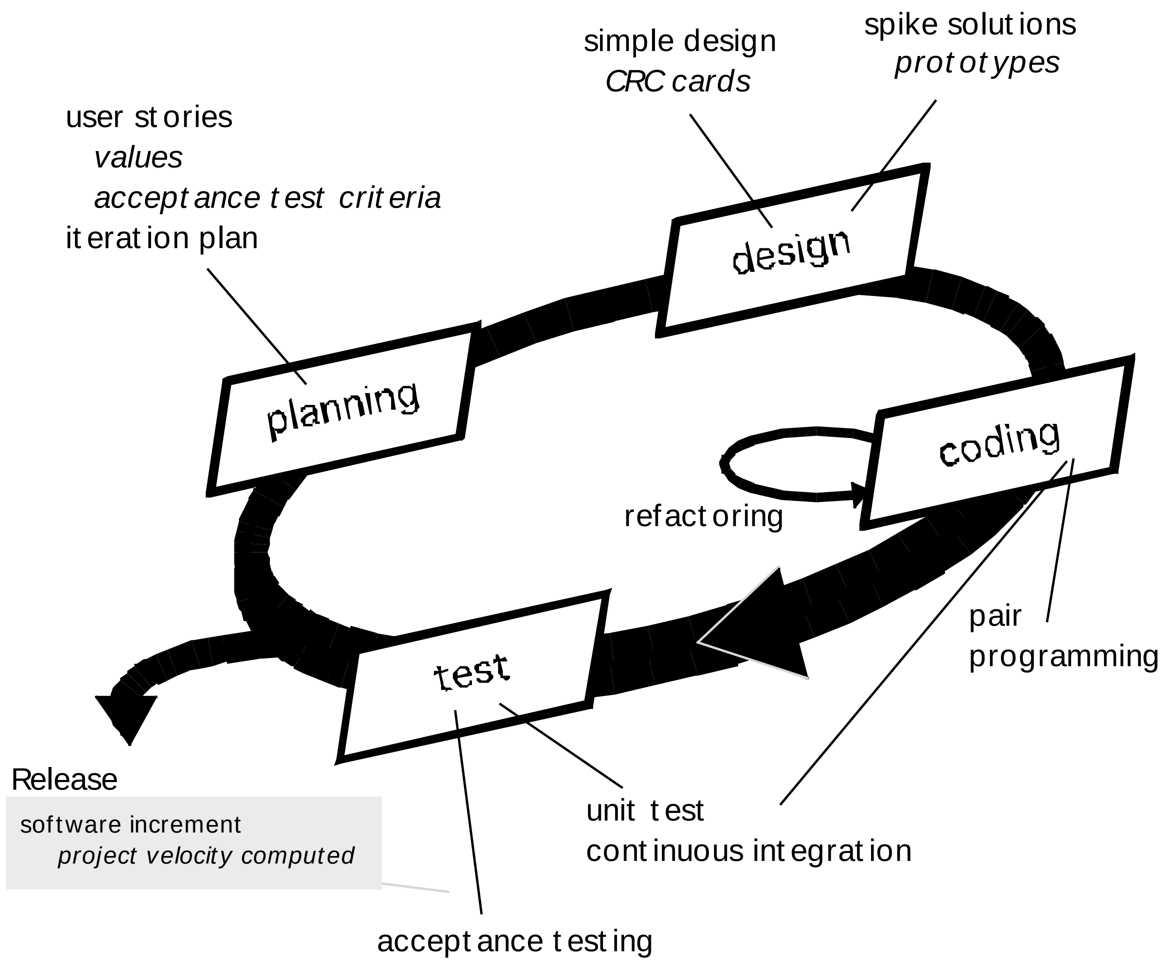
Human Factors

- the process molds to the needs of the people and team, not the other way around
- key traits must exist among the people on an agile team and the team itself:
  - Competence. (talent, skills, knowledge)
  - Common focus. (deliver a working software increment)
  - Collaboration. (peers and stakeholders)
  - Decision-making ability. (freedom to control its own destiny)
  - Fuzzy problem-solving ability.(ambiguity and constant changes, today problem may not be tomorrow's problem)
  - Mutual trust and respect.
  - Self-organization. (themselves for the work done, process for its local environment, the work schedule)

Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck in 2004. It uses an object-oriented approach.
- XP Planning
  - Begins with the listening, leads to creation of "user stories" that describes required output, features, and functionality. Customer assigns a value(i.e., a priority) to each story.
  - Agile team assesses each story and assigns a cost (development weeks. If more than 3 weeks, customer asked to split into smaller stories)
  - Working together, stories are grouped for a deliverable increment next release.
  - A commitment (stories to be included, delivery date and other project matters) is made. Three ways: 1. Either all stories will be implemented in a few weeks, 2. high priority stories first, or 3. the riskiest stories will be implemented first.
  - After the first increment "project velocity", namely number of stories implemented during the first release is used to help define subsequent delivery dates for other increments. Customers can add stories, delete existing stories, change values of an existing story, split stories as development work proceeds.

Extreme Programming (XP)



Extreme Programming (XP)

- XP Design
  - Follows the KIS principle (keep it simple) Nothing more nothing less than the story.
  - Encourage the use of CRC (class-responsibility-collaborator) cards in an object-oriented context. The only design work product of XP. They identify and organize the classes that are relevant to the current software increment.
  - For difficult design problems, suggests the creation of "spike solutions"—a design prototype for that portion is implemented and evaluated.
  - Encourages "refactoring"—an iterative refinement of the internal program design. Does not alter the external behavior yet improve the internal structure. Minimize chances of bugs. More efficient, easy to read.

Extreme Programming (XP)

- XP Coding
  - Recommends the construction of a unit test for a story before coding commences. So implementer can focus on what must be implemented to pass the test.
  - Encourages “pair programming”. Two people work together at one workstation. Real time problem solving, real time review for quality assurance. Take slightly different roles.
- XP Testing
  - All unit tests are executed daily and ideally should be automated. Regression tests are conducted to test current and previous components.
  - “Acceptance tests” are defined by the customer and executed to assess customer visible functionality

The XP Debate

- Requirements volatility: customer is an active member of XP team, changes to requirements are requested informally and frequently.
- Conflicting customer needs: different customers' needs need to be assimilated. Different vision or beyond their authority.
- Requirements are expressed informally: Use stories and acceptance tests are the only explicit manifestation of requirements. Formal models may avoid inconsistencies and errors before the system is built. Proponents said changing nature makes such models obsolete as soon as they are developed.
- Lack of formal design: XP deemphasizes the need for architectural design. Complex systems need overall structure to exhibit quality and maintainability. Proponents said incremental nature limits complexity as simplicity is a core value.

SCRUM

- Scrum is an agile process that allows us to focus on delivering the highest business value in the shortest time.
- It allows us to rapidly and repeatedly inspect actual working software (every two weeks to one month).
- The business sets the priorities. Our teams self-manage to determine the best way to deliver the highest priority features.
- Every two weeks to a month anyone can see real working software and decide to release it as is or continue to enhance for another iteration.

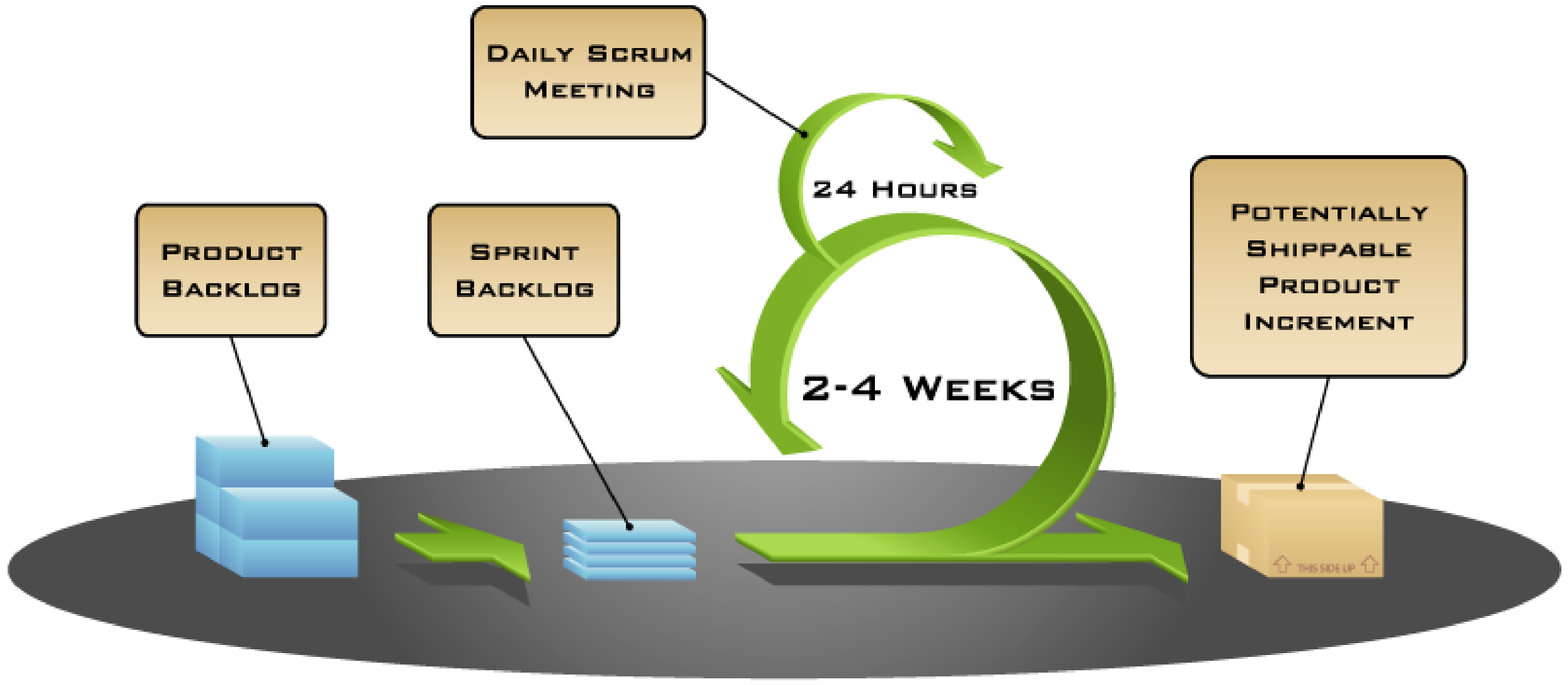
SCRUM

- A software development method Originally proposed by Schwaber and Beedle (an activity occurs during a rugby match) in early 1990.
  - Scrum—distinguishing features
  - Development work is partitioned into "packets"
  - Testing and documentation are on-going as the product is constructed
  - Work units occurs in "sprints" and is derived from a "backlog" of existing changing prioritized requirements
  - Changes are not introduced in sprints (short term but stable) but in backlog.
  - Meetings are very short (15 minutes daily) and sometimes conducted without chairs (what did you do since last meeting? What obstacles are you encountering? What do you plan to accomplish by next meeting?)
  - "demos" are delivered to the customer with the time-box allocated. May not contain all functionalities. So customers can evaluate and give feedbacks.

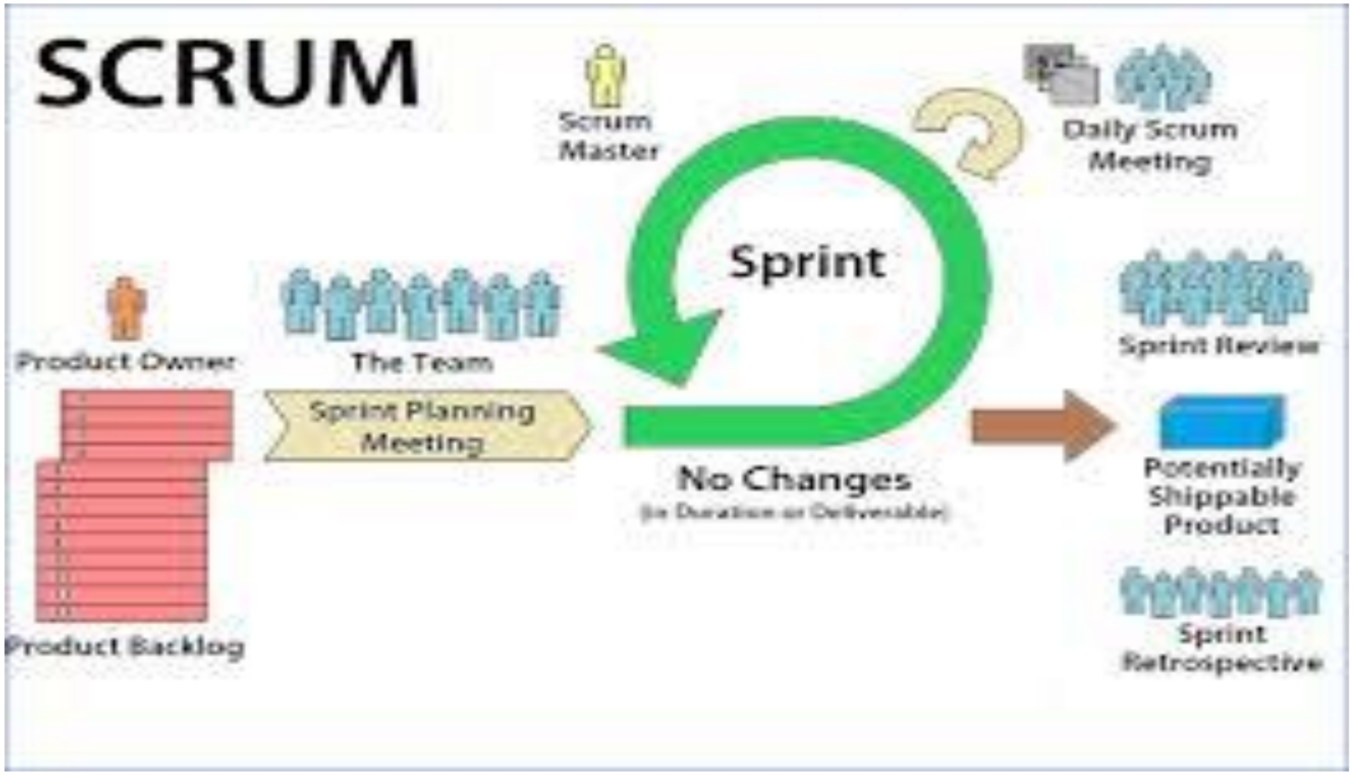
Characteristics

- Self-organizing teams
- Product progresses in a series of month-long "sprints"
- Requirements are captured as items in a list of "product backlog"
- No specific engineering practices prescribed
- Uses generative rules to create an agile environment for delivering projects
- One of the "agile processes"

How Scrum Works?



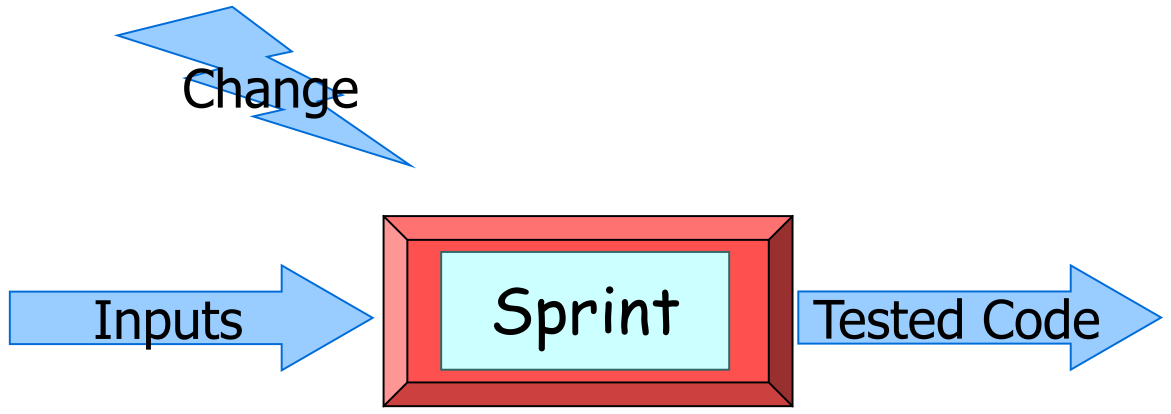
Scrum


SCRUM
Scrum Master
Daily Scrum Meeting
Sprint
Product Owner
The Team
Sprint Review
Sprint Planning Meeting
No Changes
(in Duration or Deliverable)
Potentially Shippable Product
Product Backlog
Sprint Retrospective

Sprints

- Scrum projects make progress in a series of "sprints"
    - Analogous to XP iterations
- Target duration is one month
    - +/- a week or two
    - But, a constant duration leads to a better rhythm
- Product is designed, coded, and tested during the sprint

No changes during the sprint



- Plan sprint durations around how long you can commit to keeping change out of the sprint

Scrum Framework

- Roles : Product Owner, Scrum Master, Team
- Ceremonies : Sprint Planning, Sprint Review, Sprint Retrospective, & Daily Scrum Meeting
- Artifacts : Product Backlog, Sprint Backlog, and Burndown Chart

Product Owner

- Define the features of the product
- Decide on release date and content
- Be responsible for the profitability of the product (ROI)
- Prioritize features according to market value
- Adjust features and priority every iteration, as needed
- Accept or reject work results.

The Scrum Master

- Represents management to the project
- Responsible for enacting Scrum values and practices
- Ensure that the team is fully functional and productive
- Enable close cooperation across all roles and functions
- Shield the team from external interferences

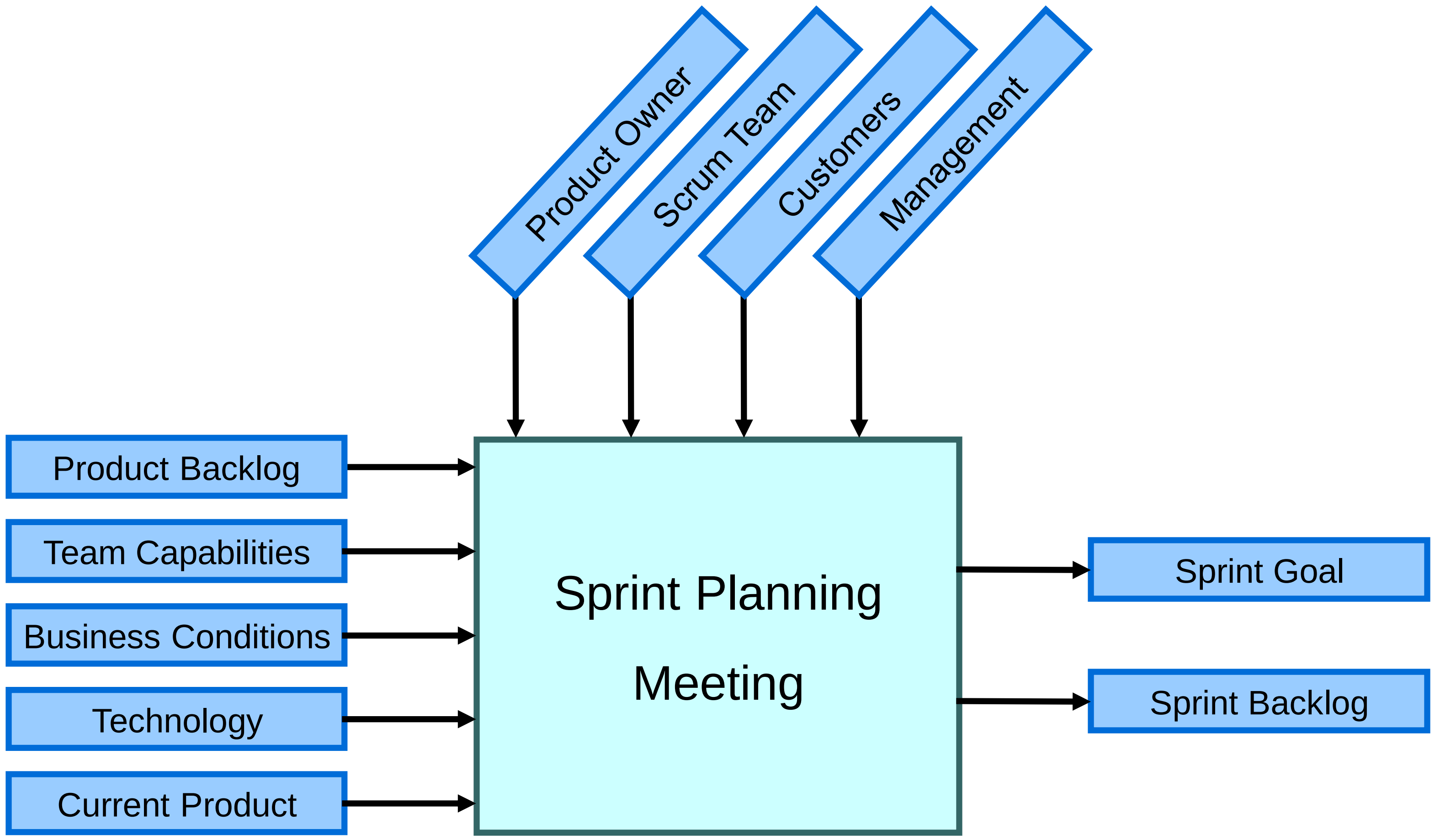
Scrum Team

- Typically 5-10 people
- Cross-functional
  - QA, Programmers, UI Designers, etc.
- Members should be full-time
  - May be exceptions (e.g., System Admin, etc.)
- Teams are self-organizing
- Membership can change only between sprints

Ceremonies

- Sprint Planning Meeting
- Sprint
- Daily Scrum
- Sprint Review Meeting

Spring Planning Meeting



Parts of Sprint Planning Meeting

- 1st Part:
  - Creating Product Backlog
  - Determining the Sprint Goal.
  - Participants: Product Owner, Scrum Master, Scrum Team
- 2nd Part:
  - Participants: Scrum Master, Scrum Team
  - Creating Sprint Backlog

Pre-Project/Kickoff Meeting

- A special form of Sprint Planning Meeting
- Meeting before the begin of the Project

Sprint

- A month-long iteration, during which is incremented a product functionality
- NO outside influence can interfere with the Scrum team during the Sprint
- Each Sprint begins with the Daily Scrum Meeting

Daily Scrum

- Parameters
  - Daily
  - 15-minutes
  - Stand-up
  - Not for problem solving
- Three questions:
  1. What did you do yesterday
  2. What will you do today?
  3. What obstacles are in your way?

Daily Scrum

- Is NOT a problem solving session
- Is NOT a way to collect information about WHO is behind the schedule
- Is a meeting in which team members make commitments to each other and to the Scrum Master
- Is a good way for a Scrum Master to track the progress of the Team

Scrum FAQs

- Why daily?
  - "How does a project get to be a year late?"
    - "One day at a time."
- Can Scrum meetings be replaced by emailed status reports?
  - No
    - Entire team sees the whole picture every day
    - Create peer pressure to do what you say you'll do

Sprint Review Meeting

- Team presents what it accomplished during the sprint
- Typically takes the form of a demo of new features or underlying architecture
- Informal
  - 2-hour prep time rule
- Participants
  - Customers
  - Management
  - Product Owner
  - Other engineers

Sprint Retrospective Meeting

- Scrum Team only
- Feedback meeting
- Three questions
  - Start
  - Stop
  - Continue
- Don’t skip for the first 5-6 sprints!!!

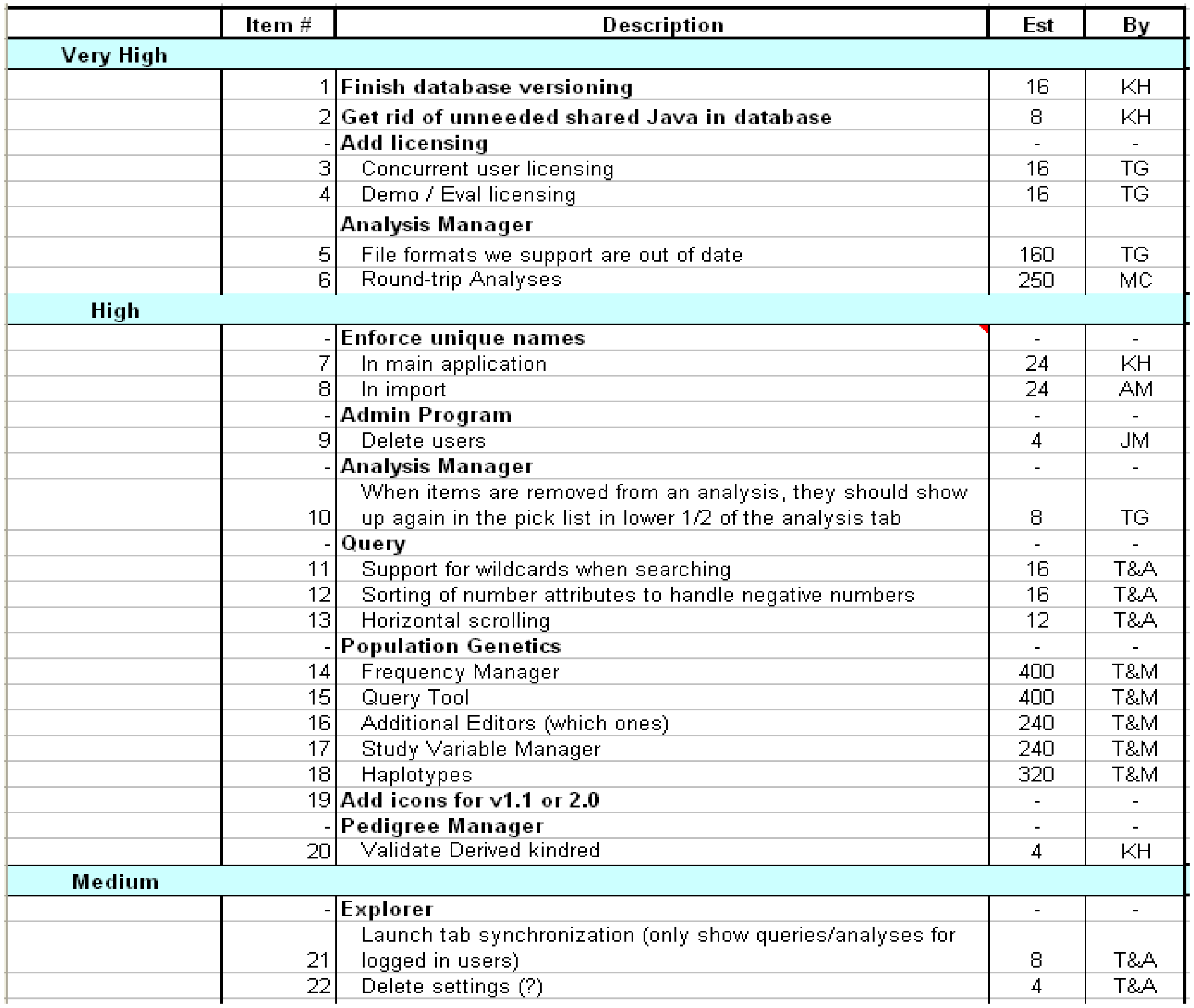
Product Backlog

- A list of all desired work on the project
  - Usually a combination of
    - story-based work (“let user search and replace”)
    - task-based work (“improve exception handling”)
- List is prioritized by the Product Owner
  - Typically a Product Manager, Marketing, Internal Customer, etc.

Product Backlog

- Requirements for a system, expressed as a prioritized list of Backlog Items
- Is managed and owned by a Product Owner
- Spreadsheet (typically)
- Usually is created during the Sprint Planning Meeting
- Can be changed and re-prioritized before each PM

Sample Product Backlog

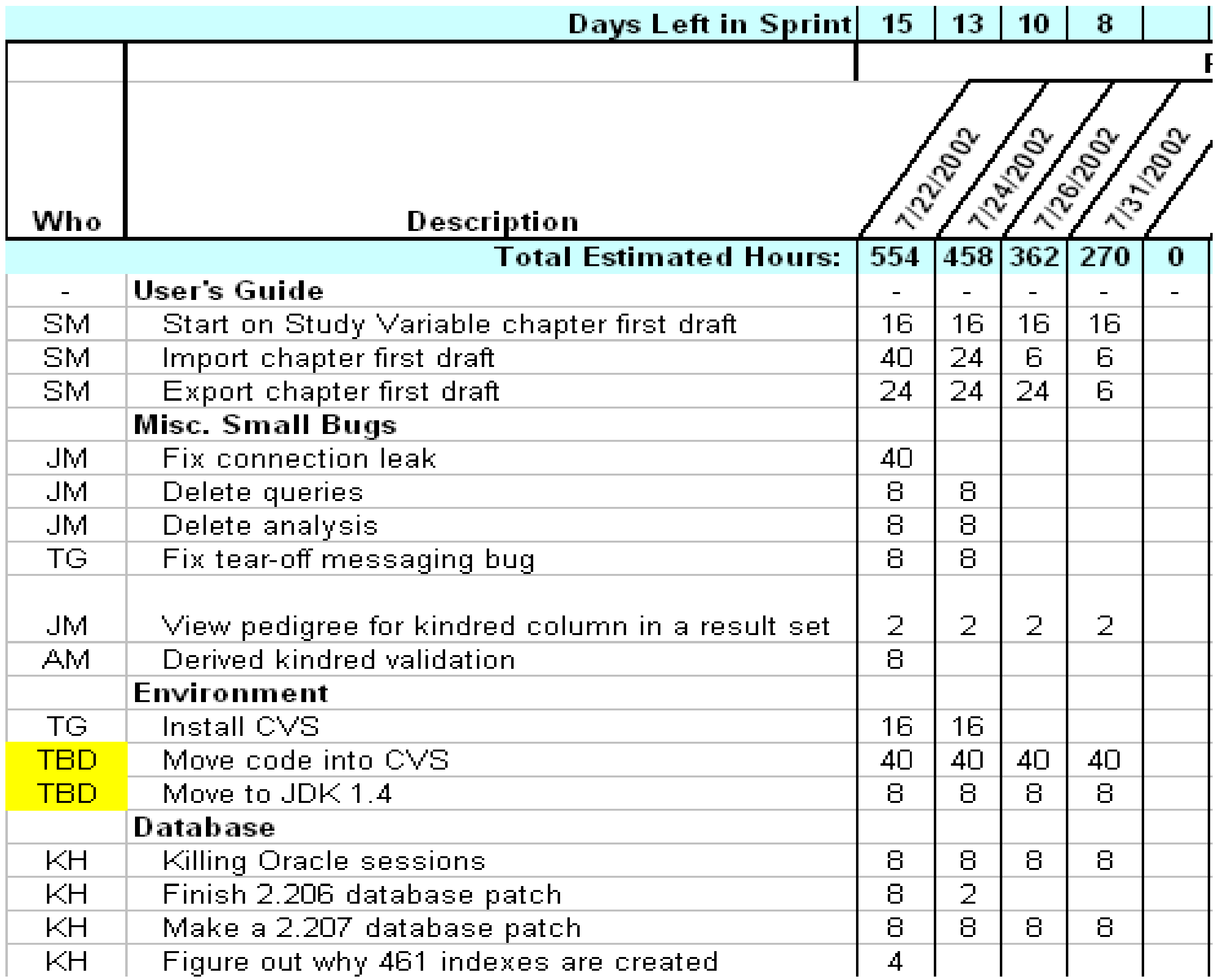


	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	**Population Genetics**	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	**Add icons for v1.1 or 2.0**	-	-
	-	**Pedigree Manager**	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	**Explorer**	-	-
	21	Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

Sprint Backlog

- A subset of Product Backlog Items, which define the work for a Sprint
- Is created ONLY by Team members
- Each Item has it's own status
- Should be updated every day
- No more than 300 tasks in the list
- If a task requires more than 16 hours, it should be broken down
- Team can add or subtract items from the list. Product Owner is not allowed to do it

Sample Sprint Backlog

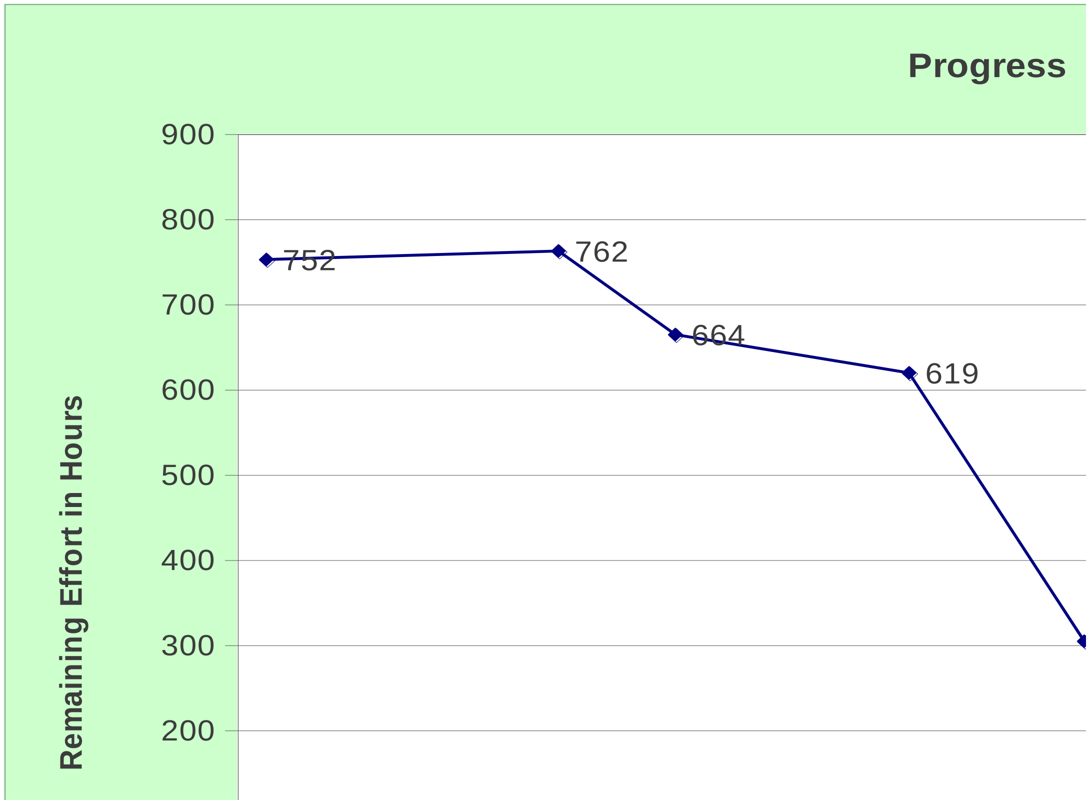


Who	Description	7/22/2002	7/24/2002	7/26/2002	7/31/2002	
	Days Left in Sprint	15	13	10	8	
	Total Estimated Hours:	554	458	362	270	0
-	**User's Guide**	-	-	-	-	-
SM	Start on Study Variable chapter first draft	16	16	16	16	
SM	Import chapter first draft	40	24	6	6	
SM	Export chapter first draft	24	24	24	6	
	Misc. Small Bugs					
JM	Fix connection leak	40				
JM	Delete queries	8	8			
JM	Delete analysis	8	8			
TG	Fix tear-off messaging bug	8	8			
JM	View pedigree for kindred column in a result set	2	2	2	2	
AM	Derived kindred validation	8				
	Environment					
TG	Install CVS	16	16			
TBD	Move code into CVS	40	40	40	40	
TBD	Move to JDK 1.4	8	8	8	8	
	Database					
KH	Killing Oracle sessions	8	8	8	8	
KH	Finish 2.206 database patch	8	2			
KH	Make a 2.207 database patch	8	8	8	8	
KH	Figure out why 461 indexes are created	4				

Sprint Burn down Chart

- Depicts the total Sprint Backlog hours remaining per day
- Shows the estimated amount of time to release
- Ideally should burn down to zero to the end of the Sprint
- Actually is not a straight line

Sprint Burn down Chart



Test Driven Development

- We produce well-designed, well-tested, and well-factored code in small, verifiable steps.
- Test-driven development, or TDD, is a rapid cycle of testing, coding, and refactoring
- If you do this correctly and in incremental steps you can reduce the defects in your system.

Test Driven Development

Benefits

- Makes finding mistakes easy.
- You express your intent twice, once with a test and another with production code.
- All these tests are checked in and become part of your continuous integration.

Test Driven Development- Challenges

- It will increase your effort. But should reduce effort at the end of delivery cycle.
- If you have legacy code extra effort and time is required to place hooks for TDD.
- The basic steps of TDD are easy to learn, but the mindset takes a while to master.
- This is a skill and requires continuous practice to get better.

Is this Pair Programming?

Pair Programming

- *We help each other succeed. This practice comes from XP.*
- When you pair, one person codes—the driver. The other person is the navigator, whose job is to think
- The driver focuses on writing syntactically correct code.
- The navigator sometimes works on understanding how the current work fits in the over-all design and sometimes thinks of the next task.
- Since we are trying to do simple design things are evolving the developers require a lot of discipline and pair programming enforces this.
- Above all it allows and forces individuals to collaborate and share knowledge.

Pair Programming- Challenges

- Pair programming can be uncomfortable in the beginning, especially if you are not used to collaborating.
- Comfort needs repeating.
- Communication issues.
- Isn't it more expensive?

Continuous Integration

- We keep our code ready to ship
- The ultimate goal of continuous integration is to be able to deploy all code.
- Although you won't release in the middle of a sprint, the point is to be technologically ready, even if you are not functionally.
- With Continuous integration, you are integrating in short cycle and thus have smaller changes to deal with as you integrate.
- Continuous integration does not make sense unless it's automated, has a short turn around time (fast builds).
- You need tests to fail or pass a build. Tests are the backbone that give you a green or a red light to take a snapshot of your build.

Continuous Integration- Challenges

- CI also requires some setup, if you don’t have one.
- Keeping build times short. This might require some serious effort and might show you the deficiency of your builds.
- And you need a good version control system – VC systems like subversion that allows atomic check-in.

Thank You