

# SOFTWARE DEVELOPMENT LIFE CYCLE

---

Instructed By: Jeremy Aschenbrenner

TIME  
FOR  
CHANGE

A light blue clock face is centered in the background. It has a white center with blue hands. The numbers 2, 4, 6, 8, 10, and 12 are visible. The text 'TIME FOR CHANGE' is overlaid in a large, white, 3D sans-serif font. 'TIME' is at the top, 'FOR' is in the middle, and 'CHANGE' is at the bottom. The clock hands are positioned at approximately 10:10.

# Common Misconception

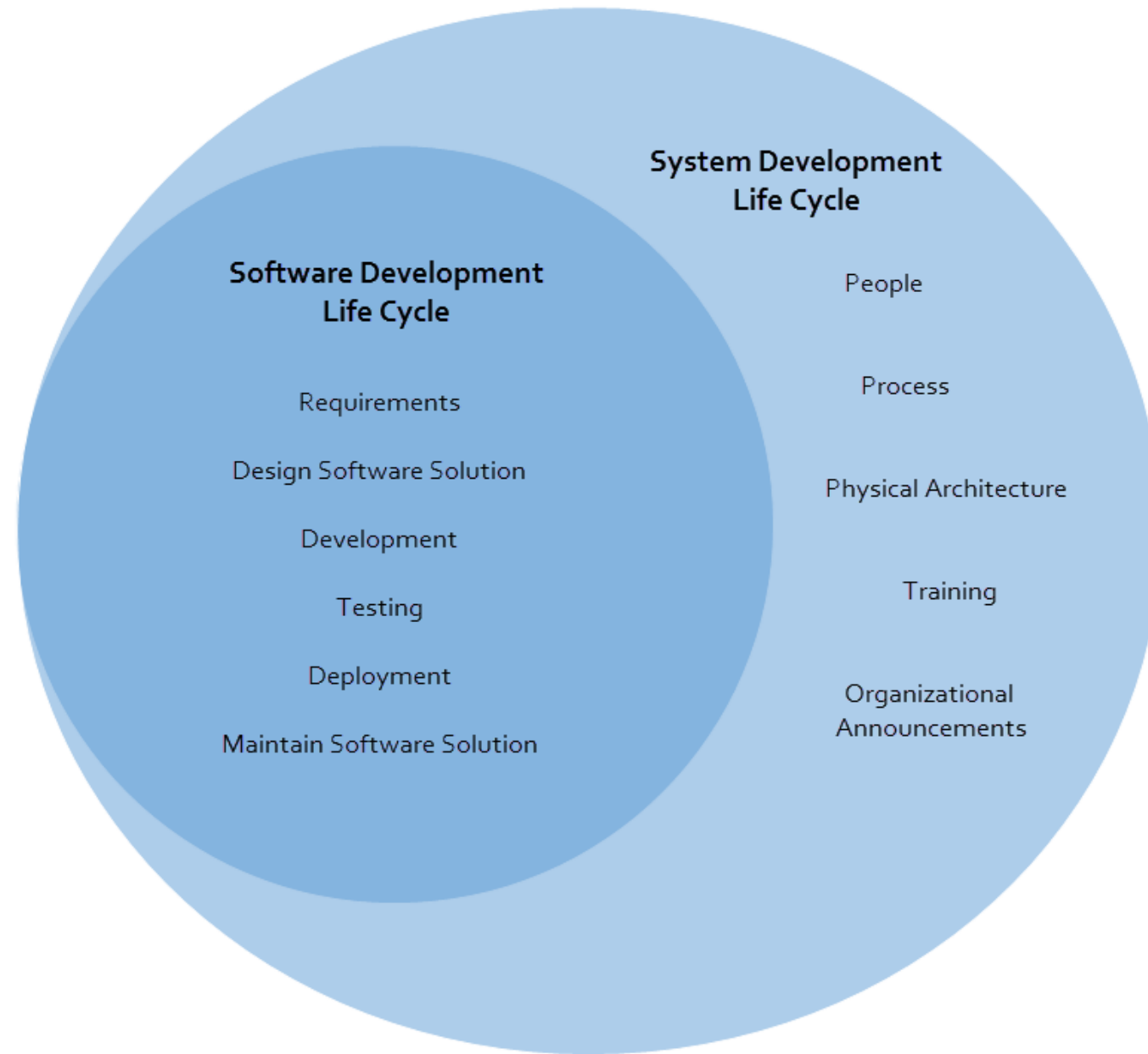
Software Development Life Cycle  
(SDLC)

vs

Systems Development Life Cycle  
(SDLC)



# SDLC VS SDLC



# Software Development Life Cycle (SDLC)

Process used to plan, create, test, and deploy an information system.



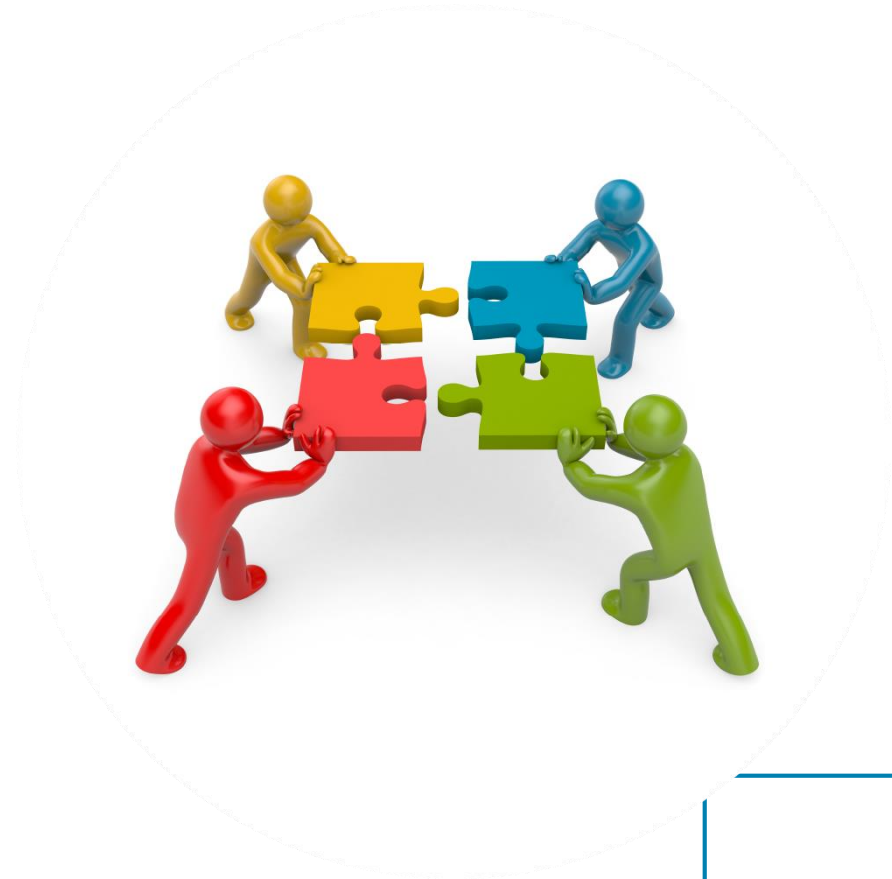
# General SDLC Steps

1. Gather Requirements
2. Design Solution
3. Development
4. Deploy
5. Maintenance



# Methodologies

- Waterfall Model
- Spiral Model
- Incremental Model
- Prototyping
- Agile
  - Scrum Model
  - Rapid Application Development (RAD)





Which methodology  
is the BEST?

Unfortunately, it's  
not that easy



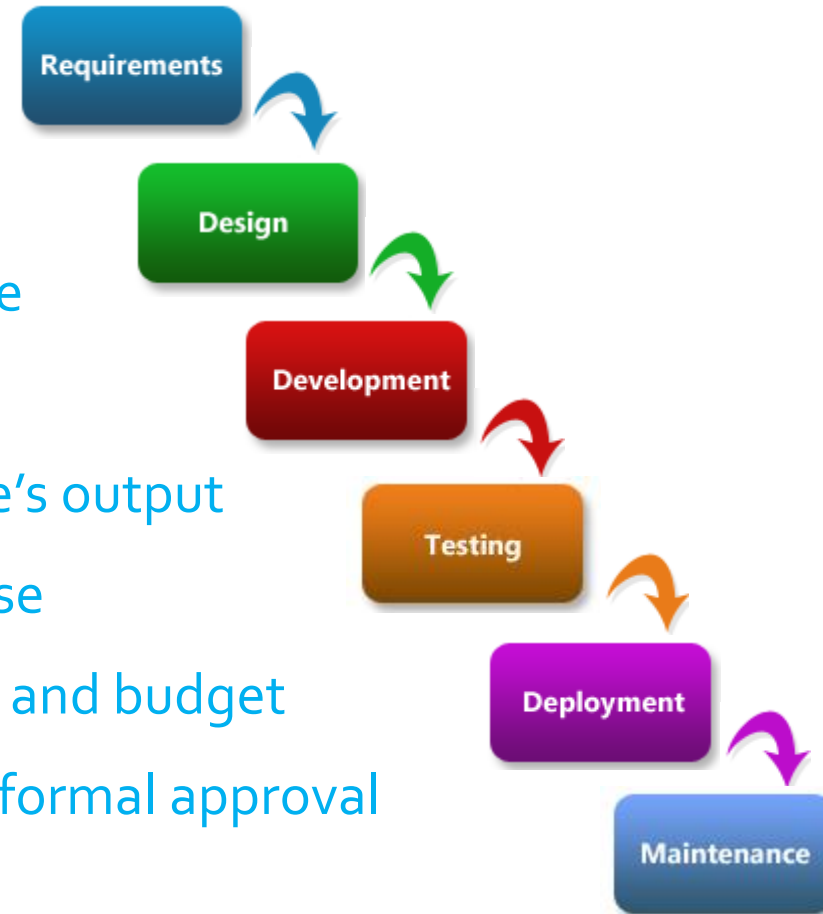


# Waterfall Model



# Waterfall Model

- Linear progression
- “Traditional” model
- Next phase begins once previous phase is complete
- Each phase begins with previous phase’s output
- No process to go back to previous phase
- Fully planned, including time schedule and budget
- Extensive written documentation and formal approval process



# Waterfall Model

Requirements

Design

Development

Testing

Deployment

Maintenance



# Waterfall Model

Requirements

- Business needs analysis
- Elicit and document requirements

Design

Development

Testing

Deployment

Maintenance



# Waterfall Model



Requirements

**Design**

Development

Testing

Deployment

Maintenance

- Design solution to meet requirements
- Determine hardware and system requirements
- Split design into segments



# Waterfall Model



Requirements

Design

**Development**

Testing

Deployment

Maintenance

- Develop/code each segment based on design
- Developers unit test each segment

# Waterfall Model



Requirements

Design

Development

**Testing**

Deployment

Maintenance

- System test each segment (QA or BA)
- User acceptance test each segment (Business)
- End-to-end test all segments together as one

# Waterfall Model



Requirements

Design

Development

Testing

**Deployment**

Maintenance

- Deploy software into production environment
- or
- Release to market

# Waterfall Model



Requirements

Design

Development

Testing

Deployment

Maintenance

- Fix bugs with patches/releases
- Enhance the product to meet changing business needs

# Waterfall Model



- + Clear expectations on schedule, budget, and resourcing needs
- + Extensive documentation helps ensure quality, reliability, and maintainability
- + Progress is easily measured
- Inflexible and cumbersome
- Long pole from project start to something tangible
- Problems are discovered at user testing
- Written documentation is never kept up-to-date, loses usefulness
- Promotes gap between the business users and the development team



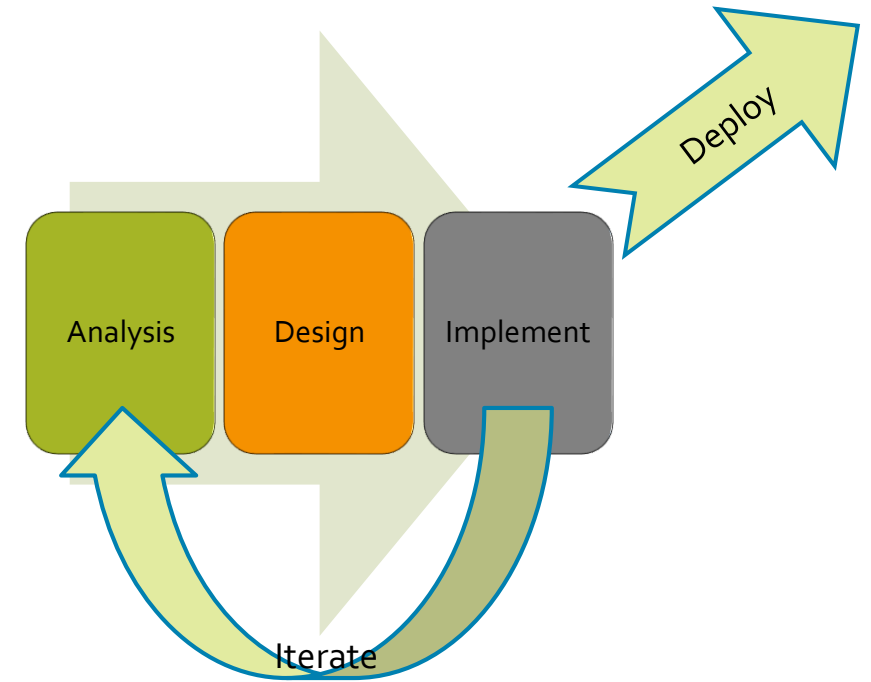
# Waterfall Model: When to Use



- Clear objectives and solution
- Large, complicated, and expensive
- No pressure for immediate return on investment (ROI) from implementation
- Project team is fully knowledgeable about the solution application
- Requirements are stable and will be unchanged through development
- Resource constraints
- Strict requirement for formal approvals at milestones

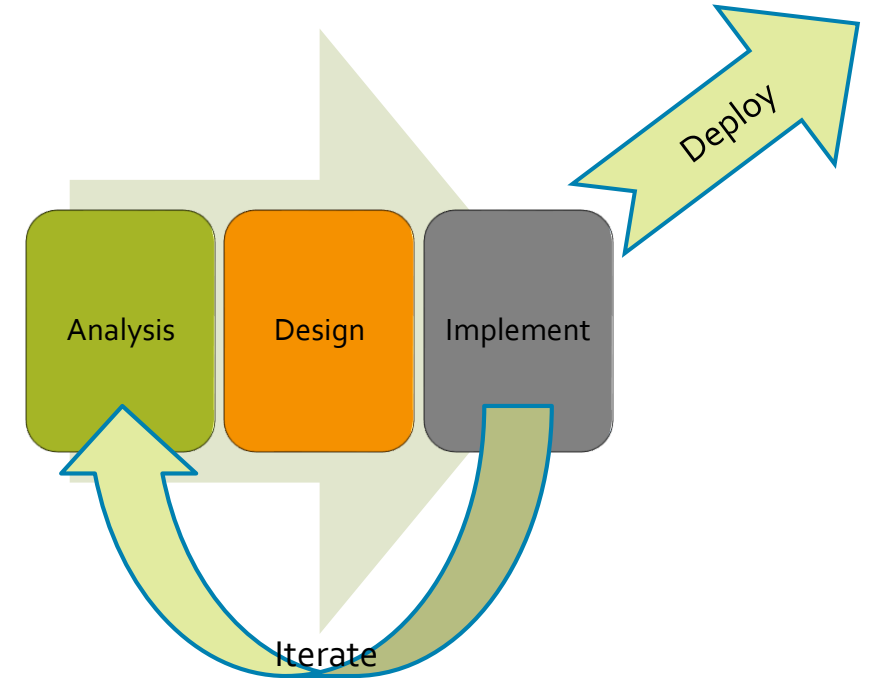


# Incremental Model



# Incremental Model

- Combination of linear and iterative approaches
- Set of concrete steps (“mini waterfalls”) that are worked through in an iterative fashion
- Allows for easier change of requirements
- Focuses on delivering customer value early in the project



# Incremental Model

- + Provides early value in the development life cycle
  - + Allows for requirement changes between iterations
  - + Problems can be detected earlier
  - + Can utilize knowledge learned in an earlier iteration
- 
- Heavy documentation, especially around interfaces with other systems
  - Can lose track of the overall business problem
  - Difficult problems tend to get pushed to future iterations



# Incremental Model: When to Use



- Large projects where requirements are not understood
- Working in unfamiliar technical arenas
- Changing requirements due to rapidly changing technology, such as leading-edge applications
- Business user can be moderately to heavily engaged
- Need functional requirements turned into something tangible quickly



# Spiral Model

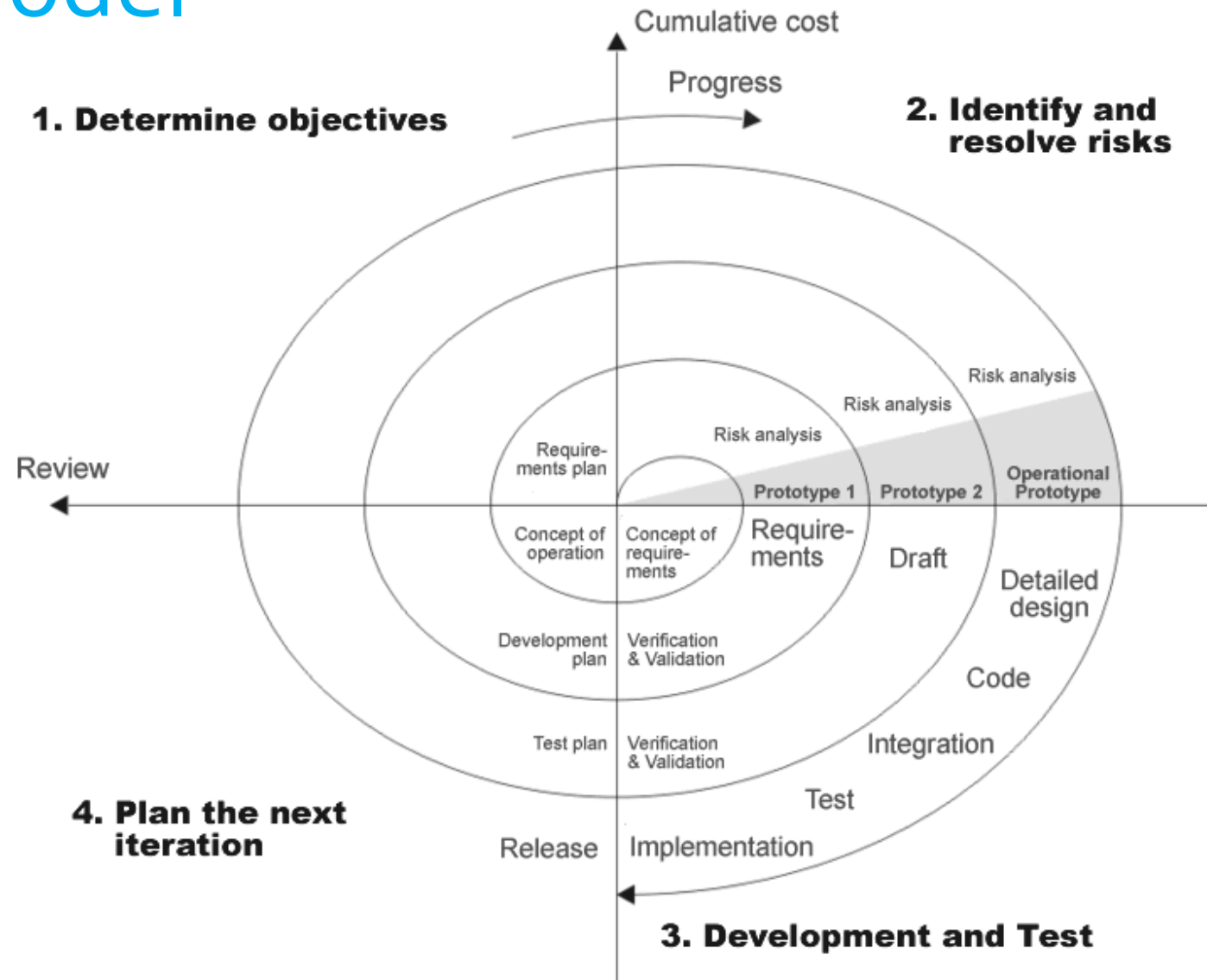


# Spiral Model

- Combination of linear and iterative
- Focuses heavily on risk assessment
- Minimized risk by breaking a project into smaller segments
- Each cycle begins with identifying stakeholders and what success looks like
- Each cycle ends with a review and a commitment



# Spiral Model



# Spiral Model

- + Accurately manages and mitigates risks
- + Issues are identified and solved early
- + Estimates near the end of the project are very accurate
- + Early involvement from developers increases successful design
- Long time to get to finished product
- High cost
- Relies on special skills to identify and mitigate risks
- Very customized solution limits reusability



# Spiral Model: When to Use

- Risk identification and mitigation are extremely important
- Medium to high risk projects
- Users are unsure of their needs
- Prototypes are needed
- Requirements are complex
- Time and cost are not as important
- Little to no experience in project's area





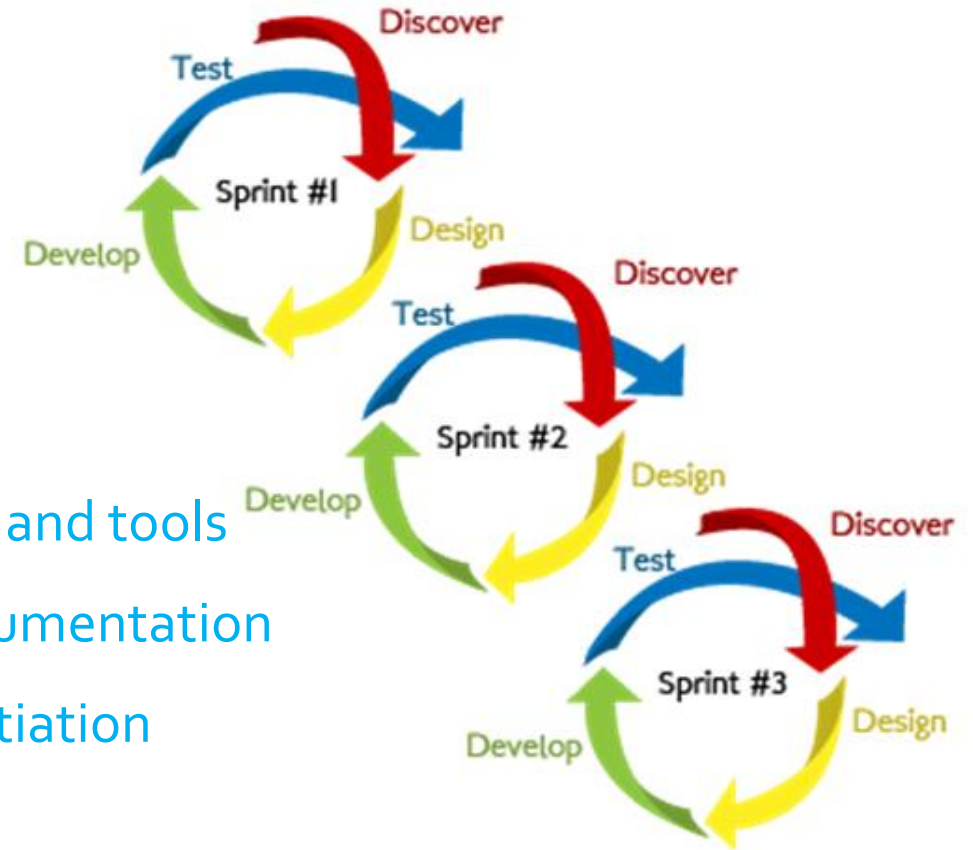
# Scrum Model (Agile)



# Scrum Model (Agile)

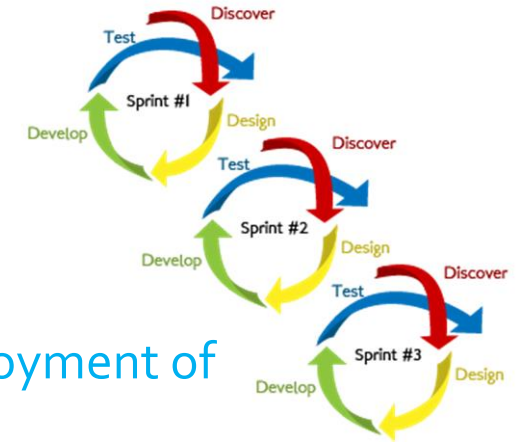
- Uses iterative approach called sprints
- Flexible and adaptable, great for the unknown
- Values individuals and interactions over processes and tools
- Values working software over comprehensive documentation
- Values customer collaboration over contract negotiation
- Values responding to change over following a plan

One of many different Agile methods including; Extreme Programming, Crystal Clear, Feature Driven, etc.



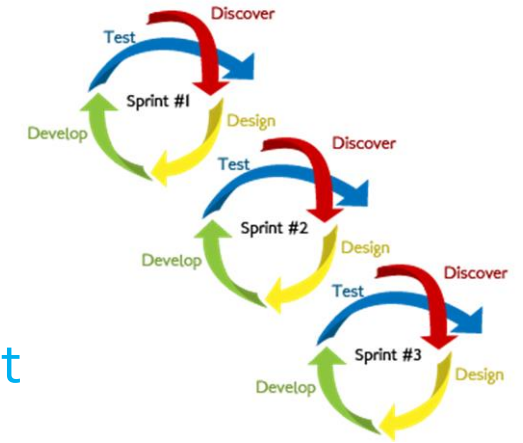
# Scrum Model (Agile)

- Project team completes sprints
  - A short iteration from analysis, development, testing and possibly to deployment of a product. (14-30 days long)
  - Uses a subset of prioritized requirements that form a sprint backlog. Requirements are derived from a product backlog
- The project team is made of Product Owner, Scrum Dev Team, and Scrum Master
  - Product Owner is responsible for the return on investment (ROI).
    - Focuses on the requirements, the “what”
  - Scrum Dev Team is cross-functional.
    - Collaborates to build the product each sprint, the “how”
  - Scrum Master is the team facilitator, but has no power.
    - Removes roadblocks, sets timeframes, and provides visibility.



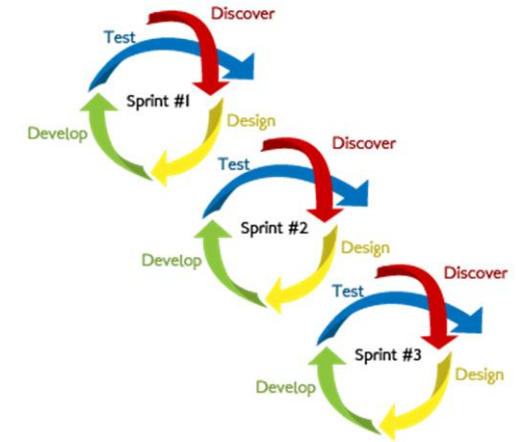
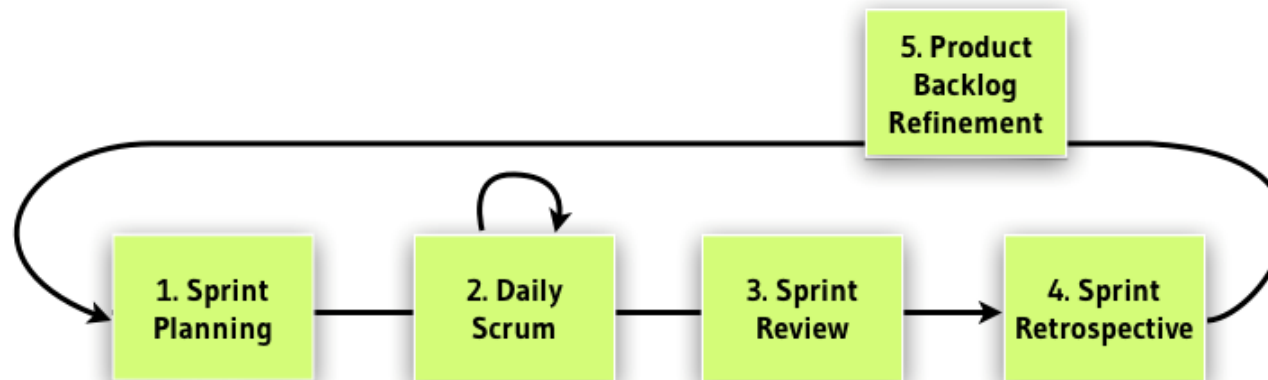
# Scrum Model (Agile): Artifacts

- Product Backlog
  - List of features that the business wants, force ranked by the Product Owner (one #1)
  - Anyone can add items, could be user stories or use cases
  - Does not contain tasks
- Sprint Backlog
  - Committed features that will be completed within the current sprint
  - Has a deadline to complete
  - Tasks are tracked as
    - Not started
    - In Progress
    - Completed



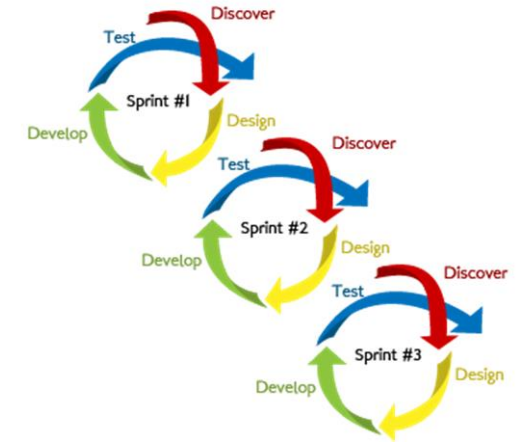
# Scrum Model (Agile): Meetings

- Sprint Planning – Commit items to the sprint backlog
- Daily Scrum – Daily, 15 minutes. Yesterday, today, roadblocks
- Sprint Review – Demonstrate product, get feedback
- Sprint Retrospective – Inspect last sprint. Lessons learned
- Product Backlog Refinement – Adjust, split, and determine dependencies of backlog items



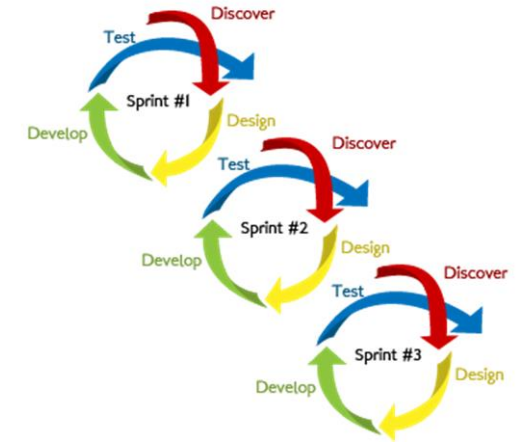
# Scrum Model (Agile)

- + Flexible and adaptable to changing requirements
  - + Gets workable product in-front of the business quicker
  - + Promotes collaboration between business and development teams
  - + Daily feedback on progress and roadblocks, stops “spinning wheels”
- 
- Leads to scope creep due to no defined end date
  - Relies on commitment of all team members
  - Challenging to initially adopt and train in an organization
  - Changing or leaving team members can have a drastically negative effect



# Scrum Model (Agile): When to Use

- The project is unpredictable and will have changing requirements
- Using or creating leading edge technology
- Organization as an experienced Scrum Master
- Business has experienced resource that can dedicate time to the project
- Pressure to produce a tangible product quickly
- Little to no concerns on length of project or budget
- Development team doesn't have resource constraints



# Rapid Application Development (RAD)



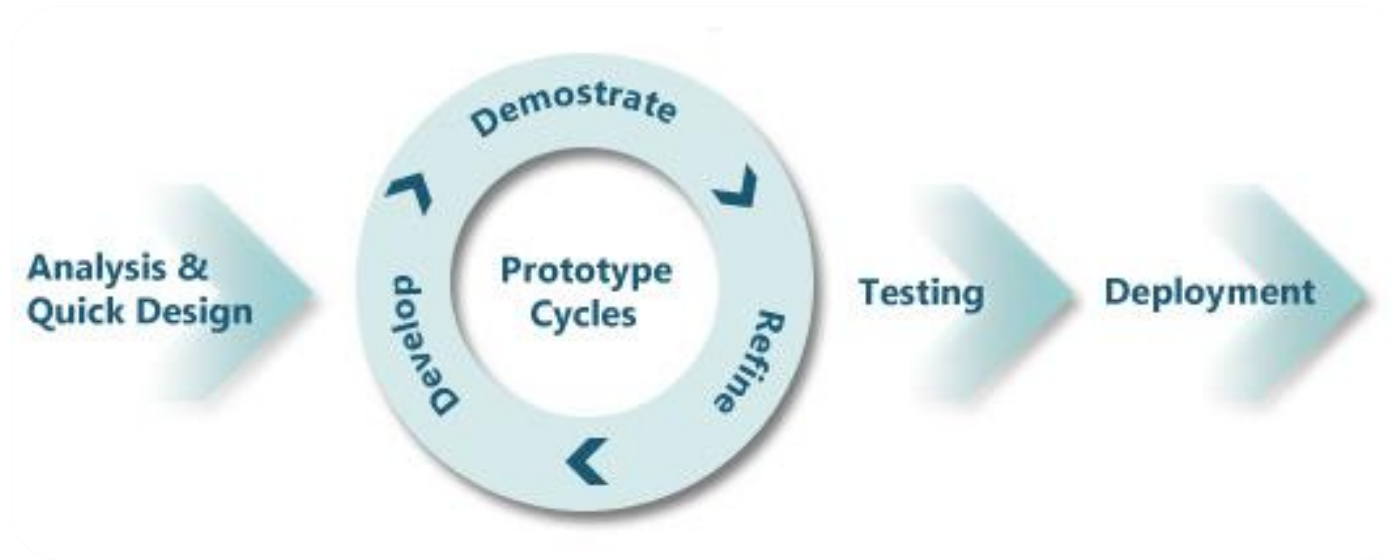


# Rapid Application Development (RAD)

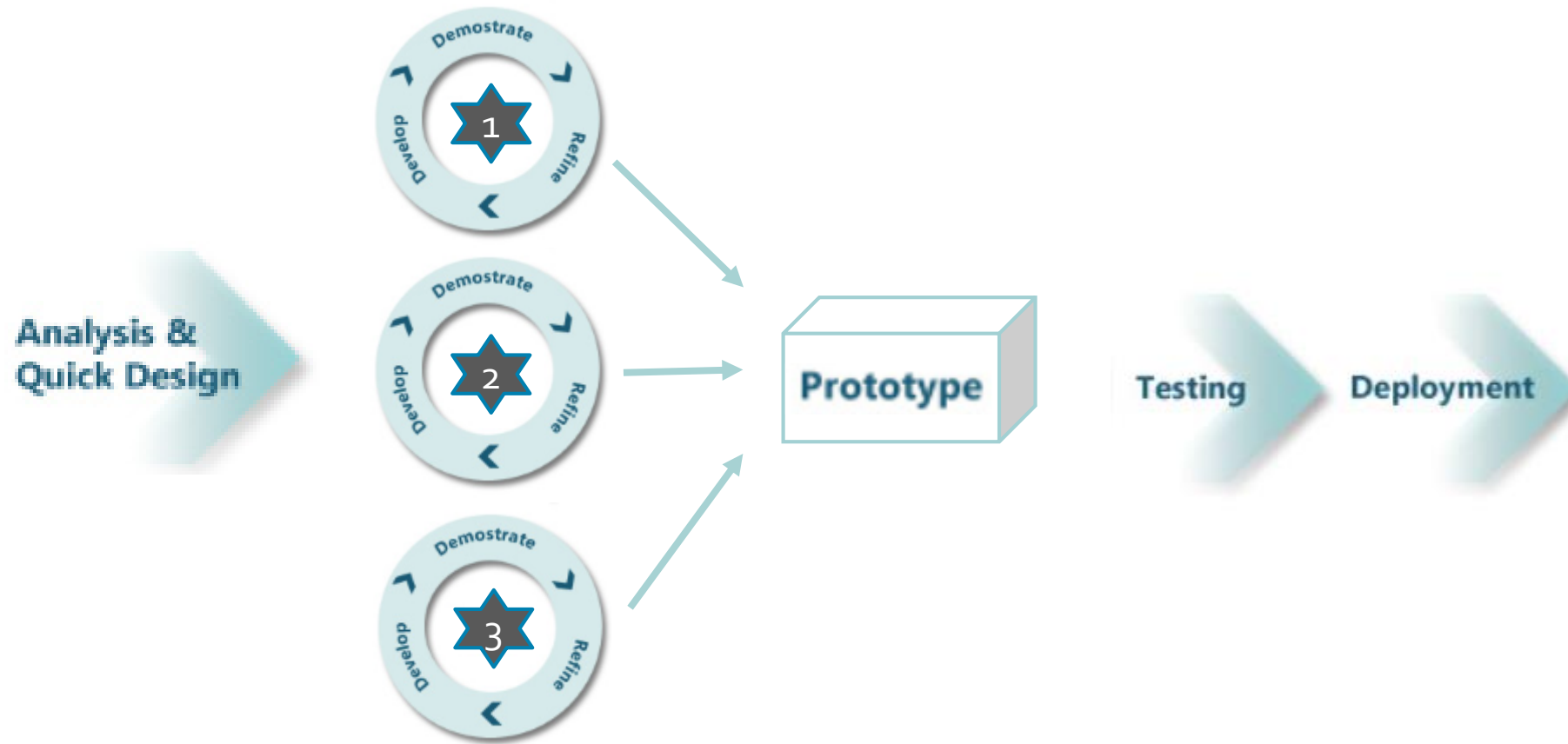
- Incremental approach
- Components and functions are developed in parallel
- Developments have tight timeframes
- The mini projects are assembled into a working prototype
- Feedback is received and prototype is refined
- Repeated until product fully meets requirements



# Rapid Application Development (RAD)



# Rapid Application Development (RAD)



# Rapid Application Development (RAD)

- + Reduced time to develop
  - + Increased reusability of components
  - + Encourages customer feedback
  - + Tackling integration early avoids later issues
- 
- Need strong team to identify business requirements
  - System must be able to be modularized
  - High dependency on modeling skills
  - Requires highly skilled developers and designers



# RAD: When to Use



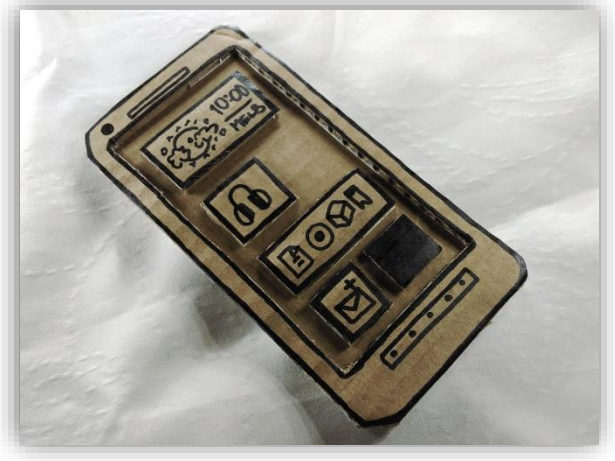
- Need a system that can be modularized and completed in 2-3 months
- High availability of quality designers for modeling
- Large budget
- Resources with high business knowledge are available to dedicate to project

# Prototyping



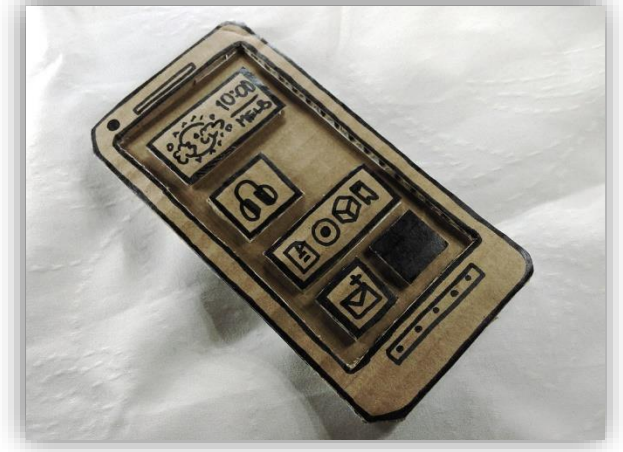
# Prototyping

- Iterative progression
- Used in conjunction with Spiral, Rapid Application Development (RAD), and Incremental models
- Breaks project into segments and creates small-scale mock-ups of the system, called prototypes
- Prototypes are iterated until they meet the requirements
- Final prototype usually discarded
- Business user is involved in the full process



# Prototype Model

- + Gives business users a visual of their wants and needs
  - + Promotes user participation and communication
  - + Allows progress even when unclear requirements
  - + Encourages innovation
- 
- Very loose approval and control process
  - Non-functional requirements are tough to identify
  - Can lead to poorly designed systems
  - False expectations thinking the system is complete from prototype





# Prototype Model: When to Use

- Project objectives are unclear
- High pressure to implement “something”
- Frequent requirement changes
- Minimal resource constraints
- No strict approval processes needed
- Innovative, flexible designs that need to accommodate future changes

