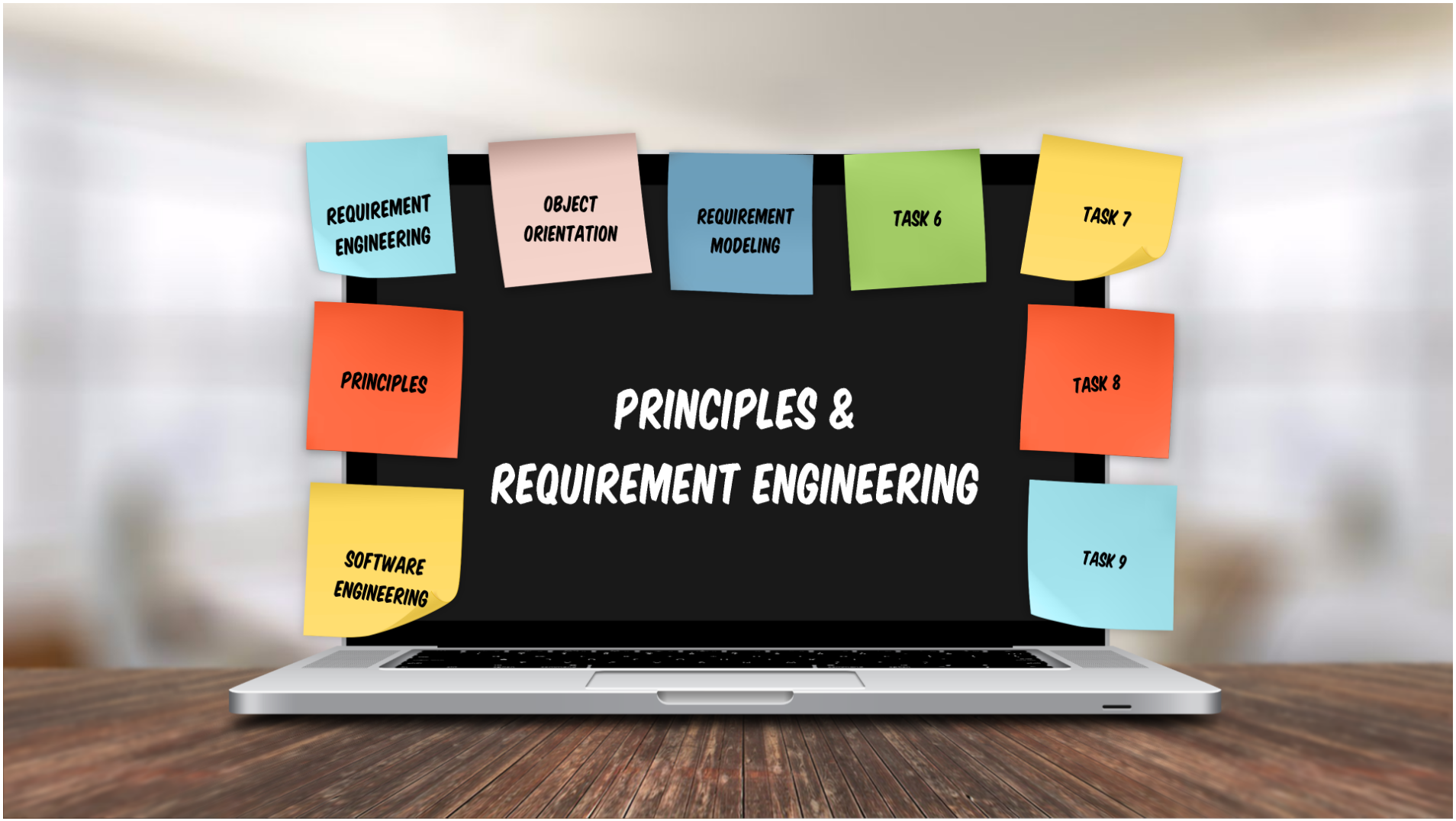
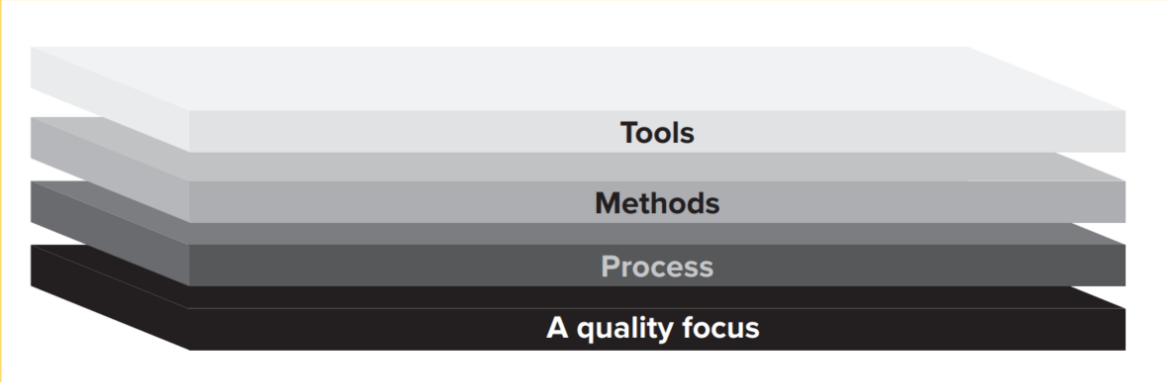
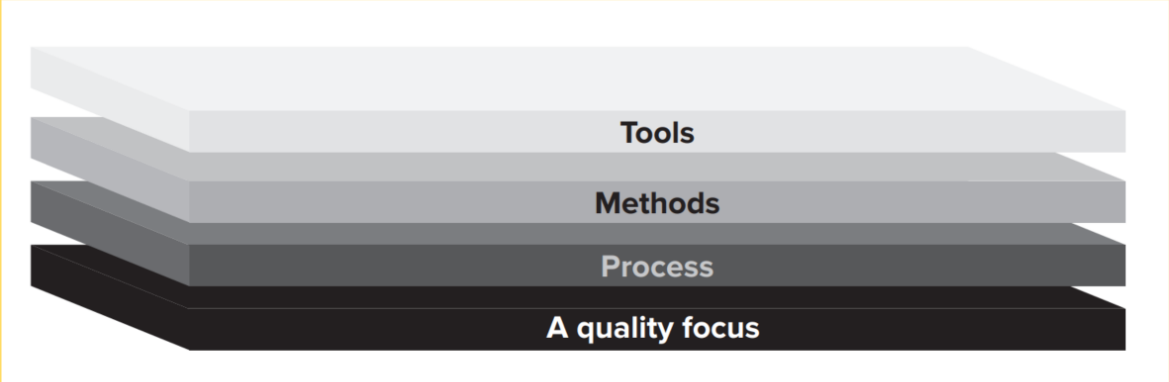

REQUIREMENT ENGINEERING
OBJECT ORIENTATION
REQUIREMENT MODELING
TASK 6
TASK 7
PRINCIPLES
PRINCIPLES &
REQUIREMENT ENGINEERING
TASK 8
SOFTWARE ENGINEERING
TASK 9

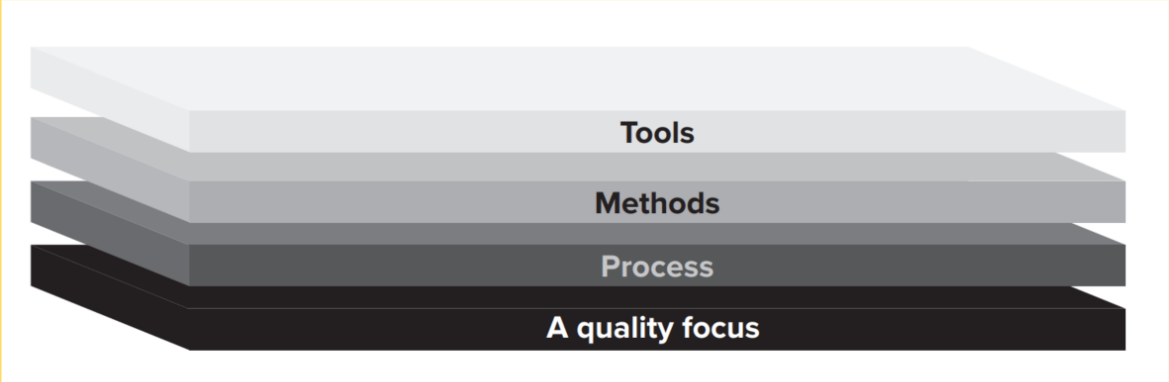
SOFTWARE ENGINEERING LAYERS



SOFTWARE ENGINEERING LAYERS

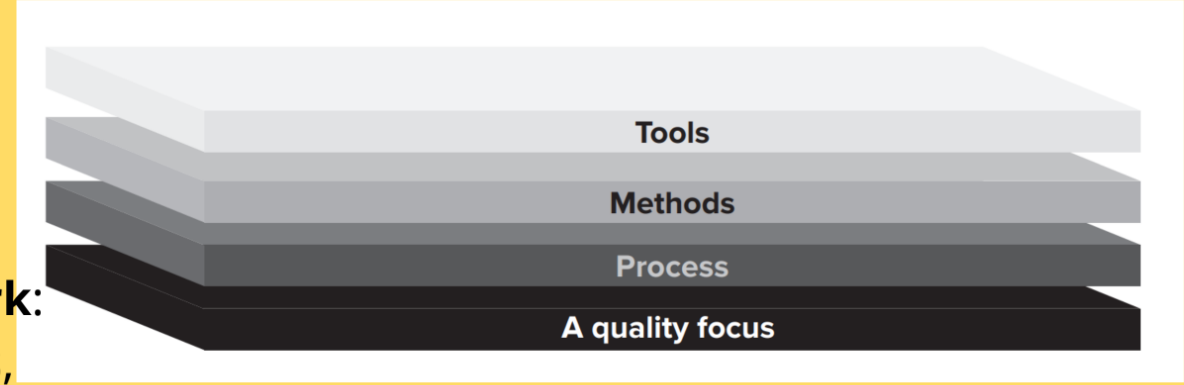


SOFTWARE ENGINEERING LAYERS



Engineering is based on **quality** promise

SOFTWARE ENGINEERING LAYERS



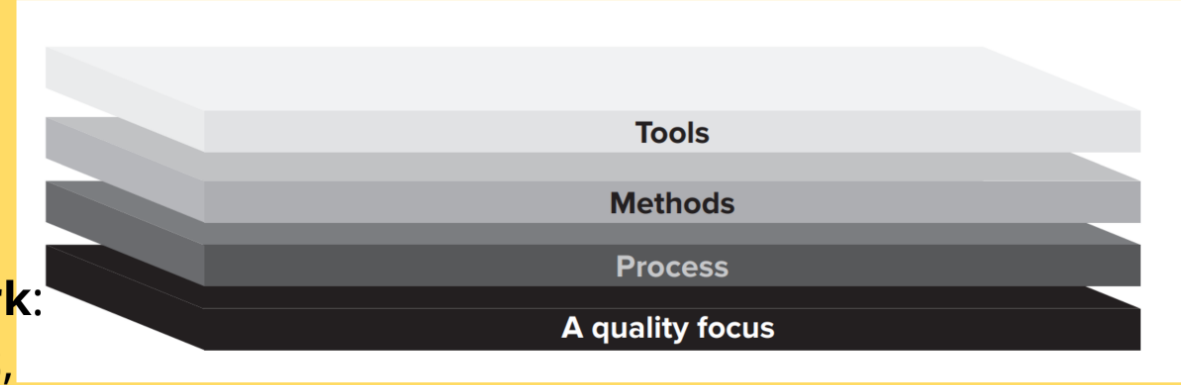
Software Process Framework: context of technical methods, work products (models, documents, data, reports, forms), milestones, change

Engineering is based on **quality** promise

SOFTWARE ENGINEERING LAYERS

Methods: how-to do tasks based on principles

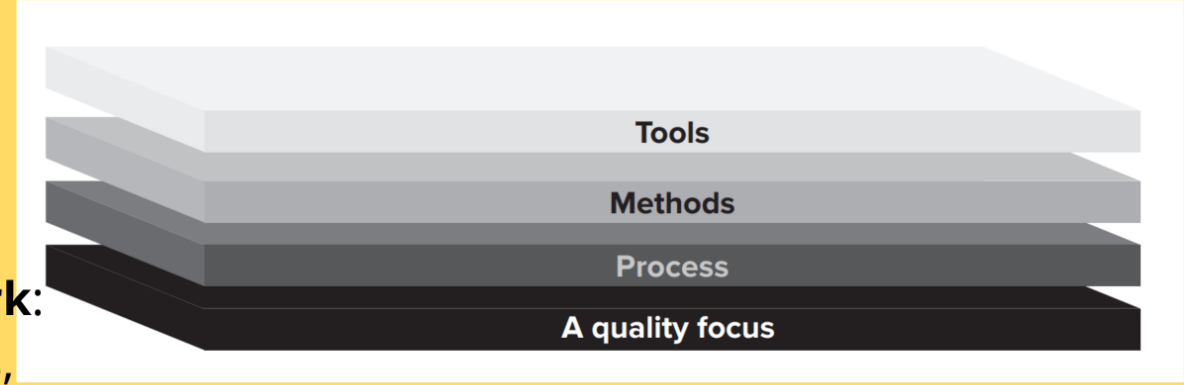
Software Process Framework: context of technical methods, work products (models, documents, data, reports, forms), milestones, change



Engineering is based on **quality** promise

SOFTWARE ENGINEERING LAYERS

Automated/semi-automated **tools**

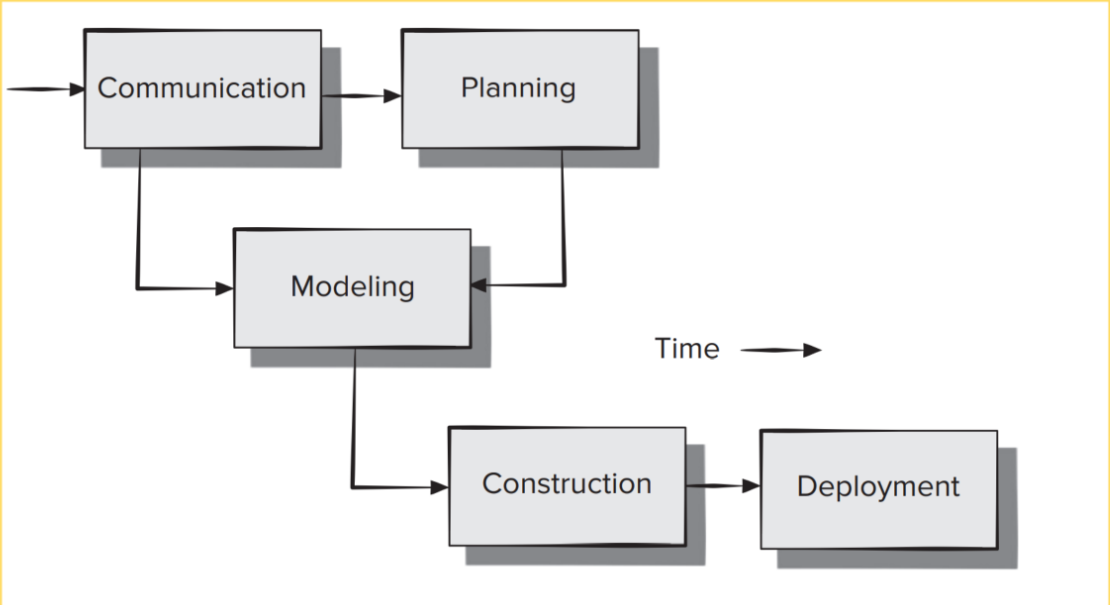


Methods: how-to do tasks based on principles

Software Process Framework: context of technical methods, work products (models, documents, data, reports, forms), milestones, change

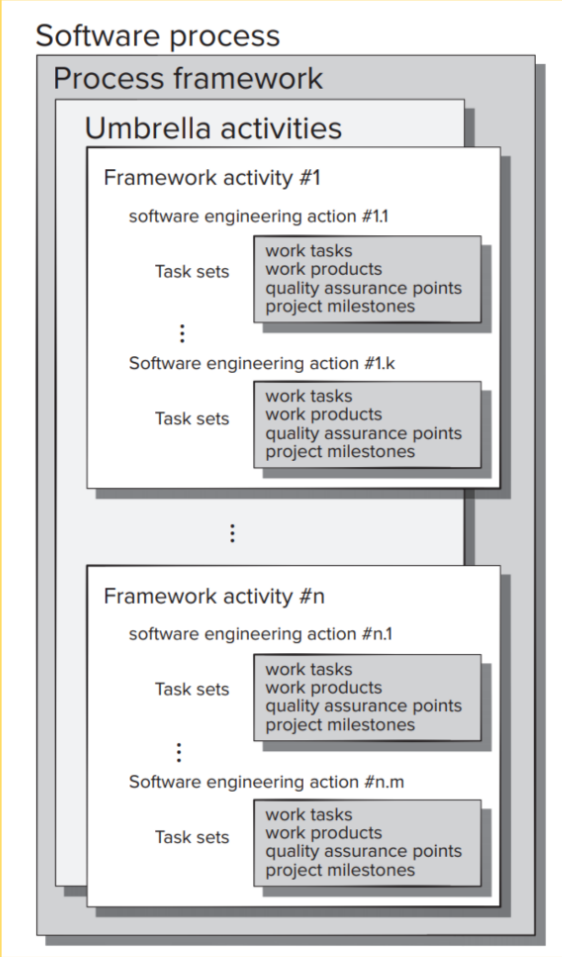
Engineering is based on **quality** promise

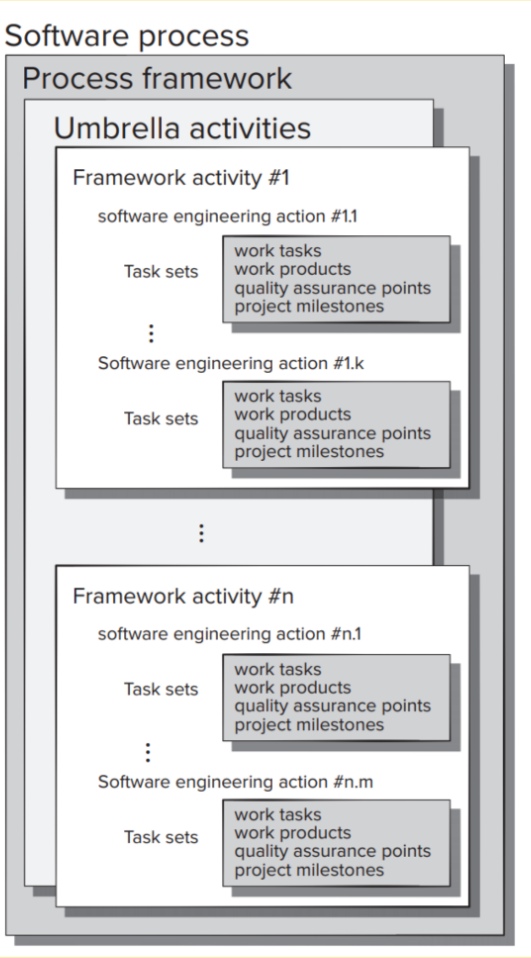
SOFTWARE PROCESS FRAMEWORK



UMBRELLA ACTIVITIES

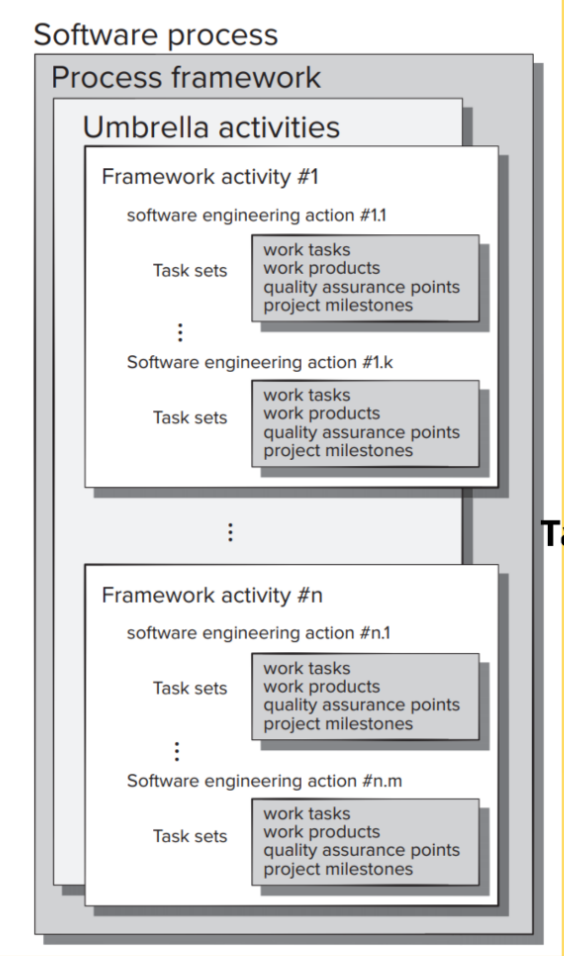
- **Software project tracking and control**: assess progress against the project plan, maintain the schedule.
- **Risk management**
- **Software quality assurance:** activities
- **Technical reviews**: assess work products to remove errors before next activity.
- **Measurement:** process, project, and product measures
- **Software configuration management**: manages change
- **Reusability management**: criteria for work product reuse (including software components), mechanisms to achieve reusable components.
- **Work product preparation and production**: models, documents, logs, forms, and lists.


Software process
Process framework
Umbrella activities
Framework activity #1
software engineering action #1.1
Task sets
work tasks
work products
quality assurance points
project milestones
Software engineering action #1.k
Task sets
work tasks
work products
quality assurance points
project milestones
Framework activity #n
software engineering action #n.1
Task sets
work tasks
work products
quality assurance points
project milestones
Software engineering action #n.m
Task sets
work tasks
work products
quality assurance points
project milestones



Communication:

- **inception**
- **elicitation**
- **elaboration**
- **negotiation**
- **specification**
- **validation**

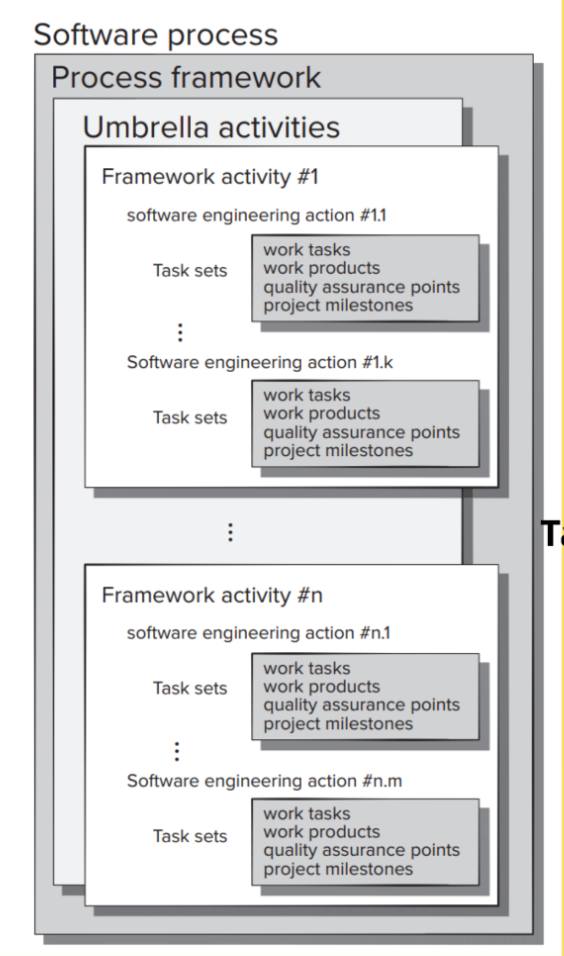


Communication:

- **inception**
- **elicitation**
- **elaboration**
- **negotiation**
- **specification**
- **validation**

Task set:

- **work tasks**
- **work products**
- **quality assurance points**
- **project milestones**



Communication:

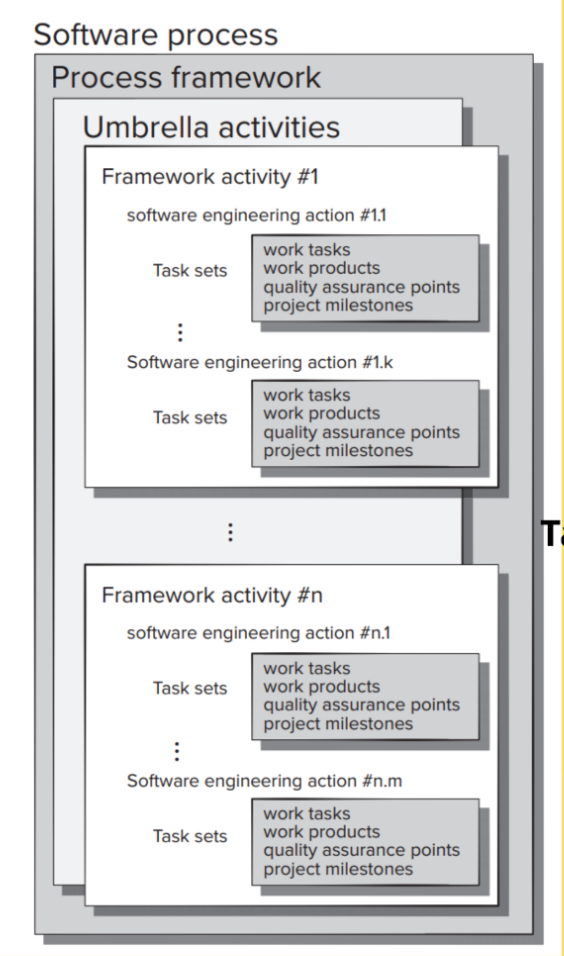
- **inception**
- **elicitation**
- **elaboration**
- **negotiation**
- **specification**
- **validation**

Elicitation task set (small):

1. **List of stakeholders**
2. **Meet all stakeholders informally**
3. **Each stakeholder list features and functions required.**
4. **Discuss requirements and build final list**
5. **Prioritize requirements**
6. **Areas of uncertainty.**

Task set:

- **work tasks**
- **work products**
- **quality assurance points**
- **project milestones**



Communication:

- **inception**
- **elicitation**
- **elaboration**
- **negotiation**
- **specification**
- **validation**

Task set:

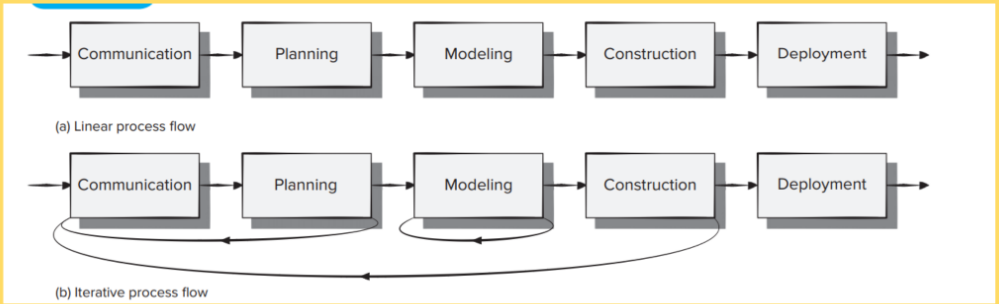
- **work tasks**
- **work products**
- **quality assurance points**
- **project milestones**

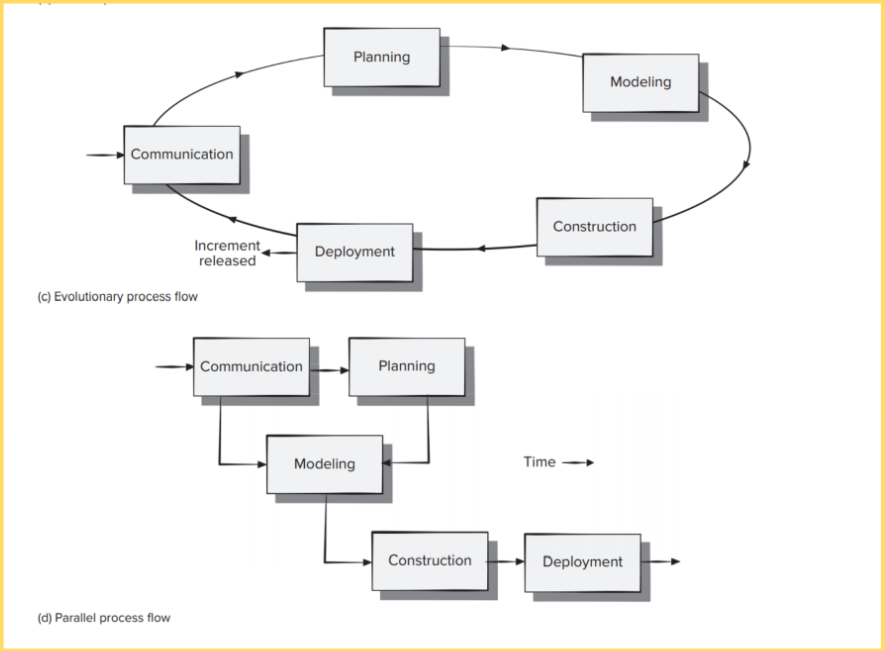
Elicitation task set (small):

1. **List of stakeholders**
2. **Meet all stakeholders informally**
3. **Each stakeholder list features and functions required.**
4. **Discuss requirements and build final list**
5. **Prioritize requirements**
6. **Areas of uncertainty.**

Elicitation task set (large & complex):

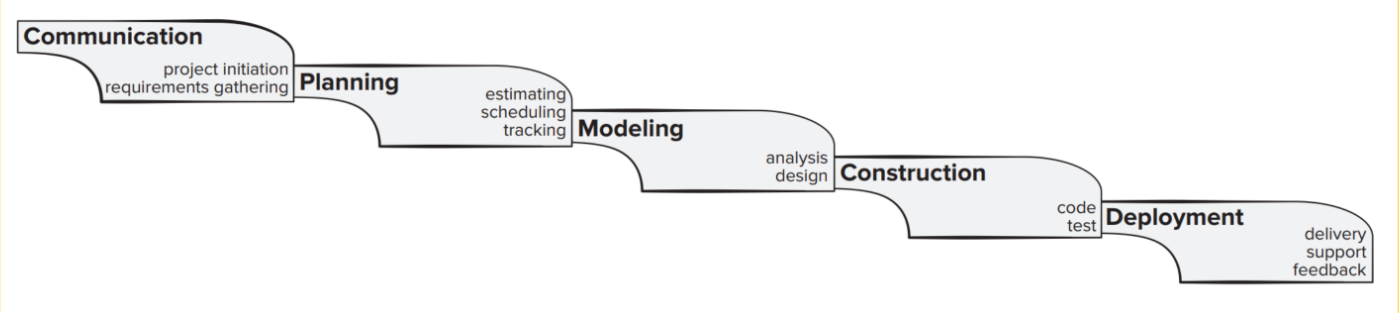
1. **List of stakeholders**
2. **Interview separately for overall wants and needs**
3. **Preliminary list of functions and features**
4. **Schedule application specification meetings**
5. **Conduct meetings**
6. **Informal user scenarios as part of each meeting**
7. **Refine user scenarios based on stakeholder feedback.**
8. **Revised list of stakeholder requirements.**
9. **Prioritize requirements with quality function deployment techniques**
10. **Package requirements for incremental delivery**
11. **Note constraints and restrictions**
12. **Discuss validation methods**



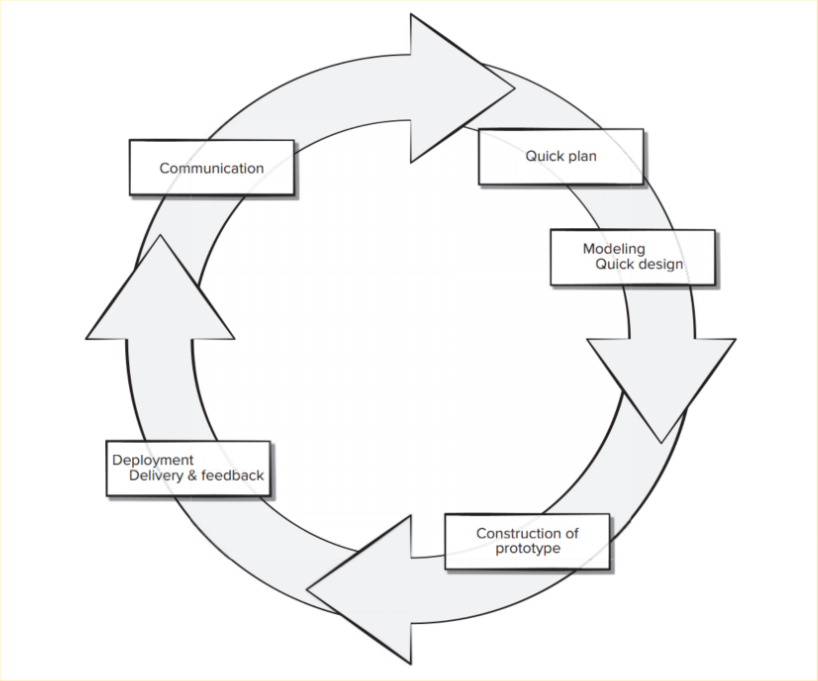


目标	属性	度量
需求质量	歧义	含糊修饰词的数量（例如：许多、大量、与人友好）
	完备性	TBA 和 TBD 的数量
	可理解性	节 / 小节的数量
	易变性	每项需求变更的数量 变更所需要的时间（通过活动）
	可追溯性	不能追溯到设计 / 代码的需求数
	模型清晰性	UML 模型数 每个模型中描述文字的页数 UML 错误数
设计质量	体系结构完整性	是否存在现成的体系结构模型
	构件完备性	追溯到结构模型的构件数 过程设计的复杂性
	接口复杂性	挑选一个典型功能或内容的平均数 布局合理性
	模式	使用的模式数量
代码质量	复杂性	环路复杂性
	可维护性	设计要素（第 23 章）
	可理解性	内部注释的百分比 变量命名约定
	可重用性	可重用构件的百分比
	文档	可读性指数
质量控制效率	资源分配	每个活动花费的人员时间百分比
	完成率	实际完成时间与预算完成时间之比
	评审效率	参见评审度量
	测试效率	发现的错误及关键性问题数 改正一个错误所需的工作量 错误的根源

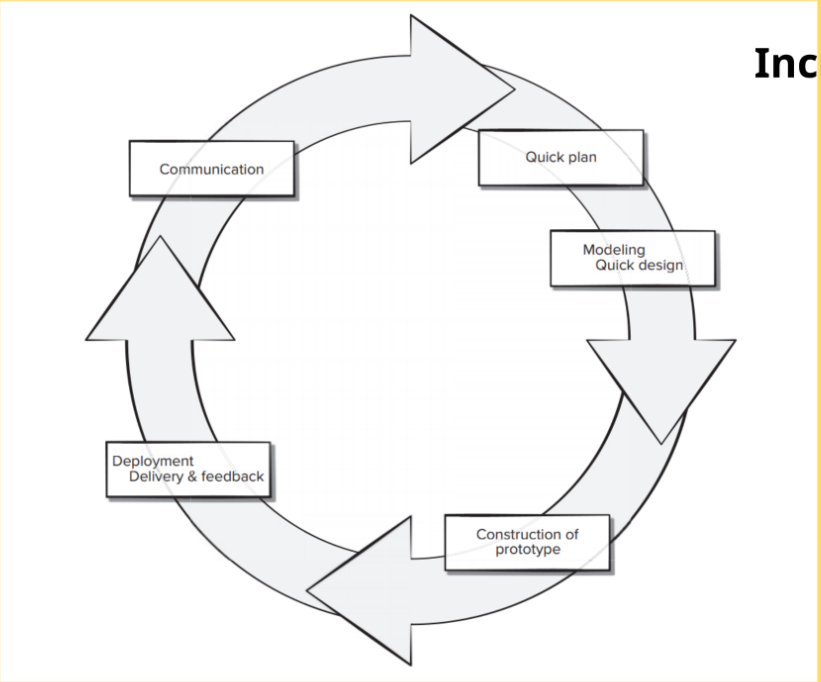
LINEAR PROCESS MODEL



PROTOTYPING PROCESS MODEL



PROTOTYPING PROCESS MODEL



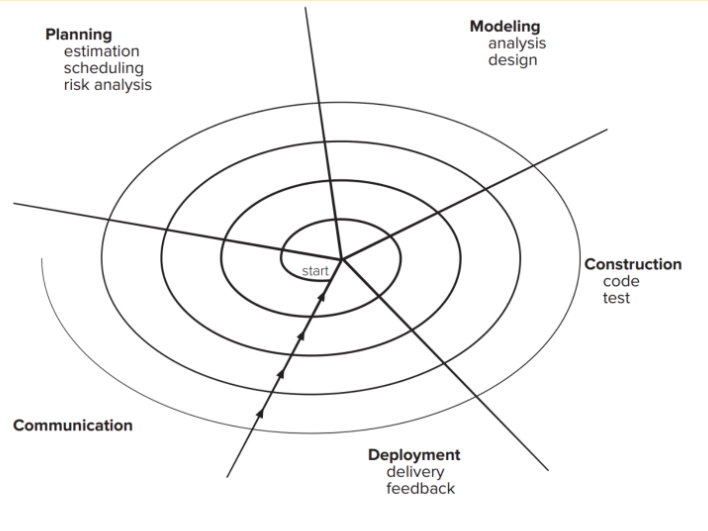
Incremental:

1. Sync a mobile phone with the fitness device and display the current data

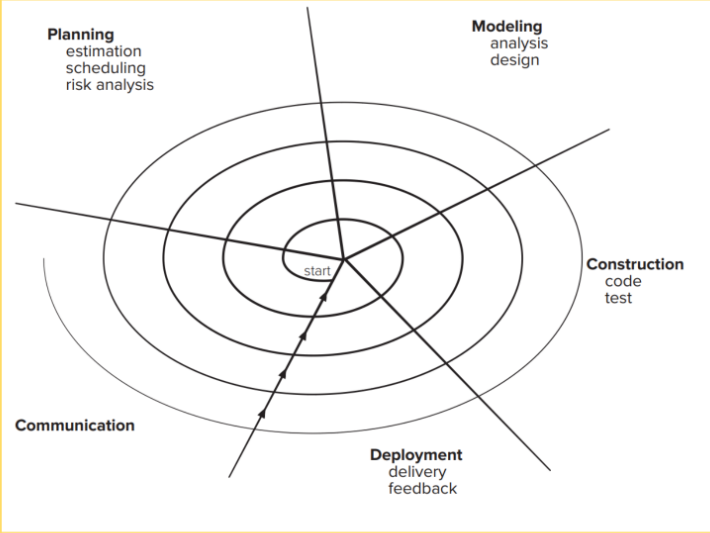
2. Set goals and store the fitness device data on the cloud (UI)

3. Social media integration to set fitness goals and share progress with a set of friends

EVOLUTIONARY PROCESS MODEL



EVOLUTIONARY PROCESS MODEL



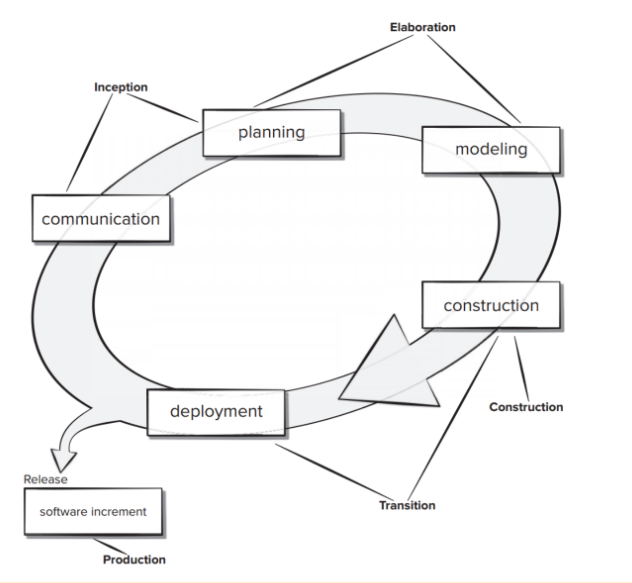
Iterative:

1. **Product specification**
2. **Prototypes**
3. **Latter versions with richer features**

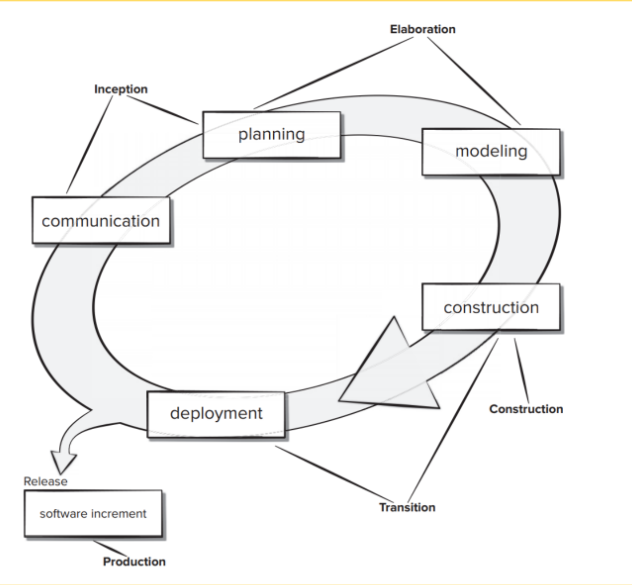
Life cycle:

1. **Replanning, changes <- feedback**
2. **Iterative and active for the full life cycle, e.g. extends to new products**
3. **Changes at the entry of each iteration**
4. **Based on professional risk management**

UNIFIED PROCESS MODEL



UNIFIED PROCESS MODEL



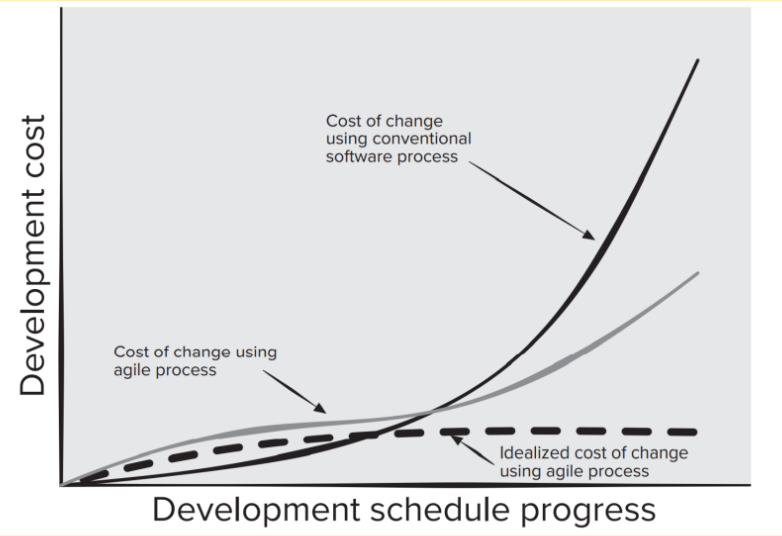
- Inception
  - Preliminary use cases - fundamental business requirements
  - Planning (increments) - resources, major risks, preliminary schedule
- Elaboration
  - Refines and expands the preliminary use cases
  - Architectural baseline: use case, analysis, design, implementation, and deployment model (not prototype)
  - Modify plans at this time
- Construction
  - Implement all necessary and required features and functions (increment)
  - Design and execute unit tests
  - Integration: component assembly, integration testing
  - Acceptance tests (prior to the beginning of next phase) <- use cases
- Transition
  - Beta testing - software and documentation given to end users
  - User feedback: defects, necessary changes
  - Software increment -> software release
- Production
  - Monitor software use
  - Infrastructure: support for the operating environment
  - Submit & evaluate defect reports & requests for changes

These phases could be concurrent!

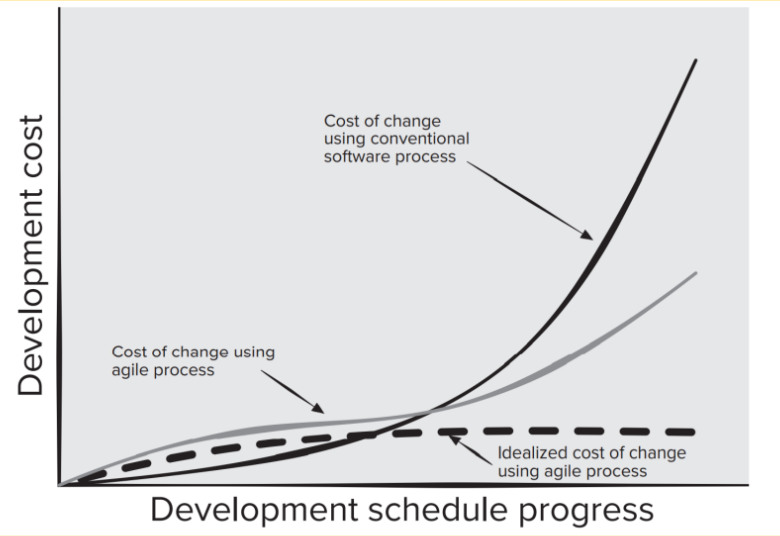
Waterfall pros	It is easy to understand and plan. It works for well-understood small projects. Analysis and testing are straightfoward.
Waterfall cons	It does not accommodate change well. Testing occurs late in the process. Customer approval is at the end.
Prototyping pros	There is a reduced impact of requirement changes. The customer is involved early and often. It works well for small projects. There is reduced likelihood of product rejection.
Prototyping cons	Customer involvement may cause delays. There may be a temptation to “ship” a prototype Work is lost in a throwaway prototype. It is hard to plan and manage.

Spiral pros	There is continuous customer involvement. Development risks are managed. It is suitable for large, complex projects. It works well for extensible products.
Spiral cons	Risk analysis failures can doom the project. The project may be hard to manage. It requires an expert development team.
Unified Process pros	Quality documentation is emphasized. There is continuous customer involvement. It accommodates requirements changes. It works well for maintenance projects.
Unified Process cons	Use cases are not always precise. It has tricky software increment integration. Overlapping phases can cause problems. It requires an expert development team.

AGILE DEVELOPMENT



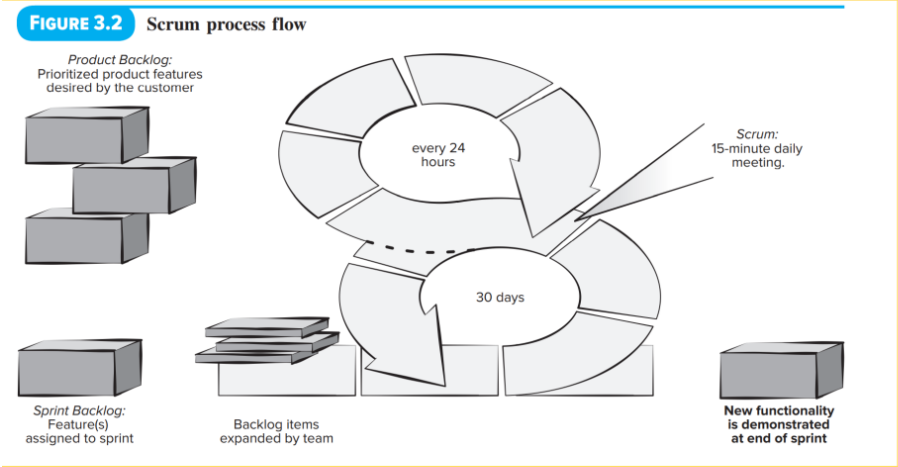
AGILE DEVELOPMENT



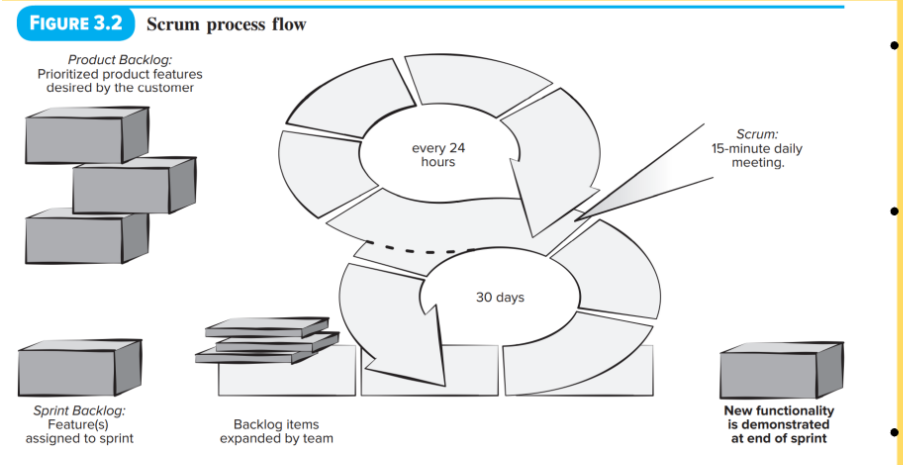
- **Hard to predict changes:**
  - **requirements**
  - **priorities**
- **Hard to predict:**
  - **how much design is necessary before using construction**
  - **Analysis, design, construction, and testing**

Adaptable, incremental & change!

AGILE DEVELOPMENT:SCRUM



AGILE DEVELOPMENT:SCRUM

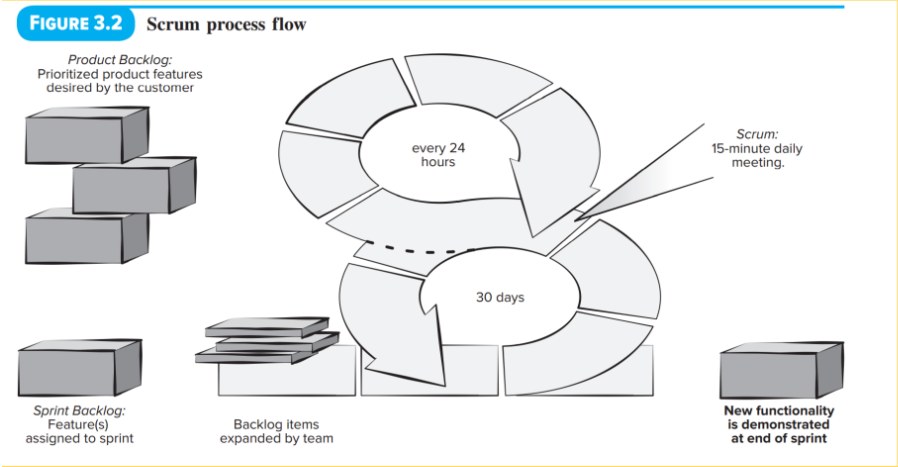


- Principal artifacts:
  - product backlog: prioritized requirements/ features (add anytime with approval)
  - sprint backlog: subset of product backlog items in current sprint
    - sprint: 2-4 weeks (decided by owner)
  - code increment
    - increment: union of backlog items in previous & current sprint

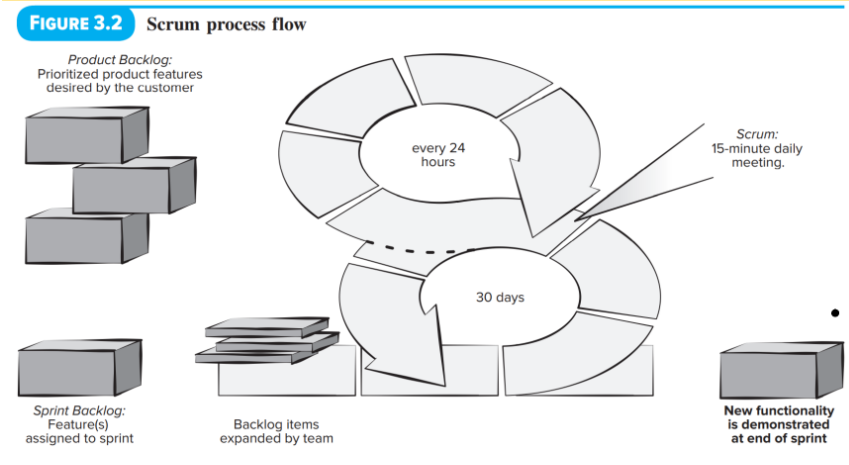
- Sprint Planning Meeting

  1. Develop the items product backlog
  2. Increment in the upcoming sprint (owner)
  3. Select items to move from to the sprint backlog (masters & development)
  4. Role allocation

AGILE DEVELOPMENT: SCRUM

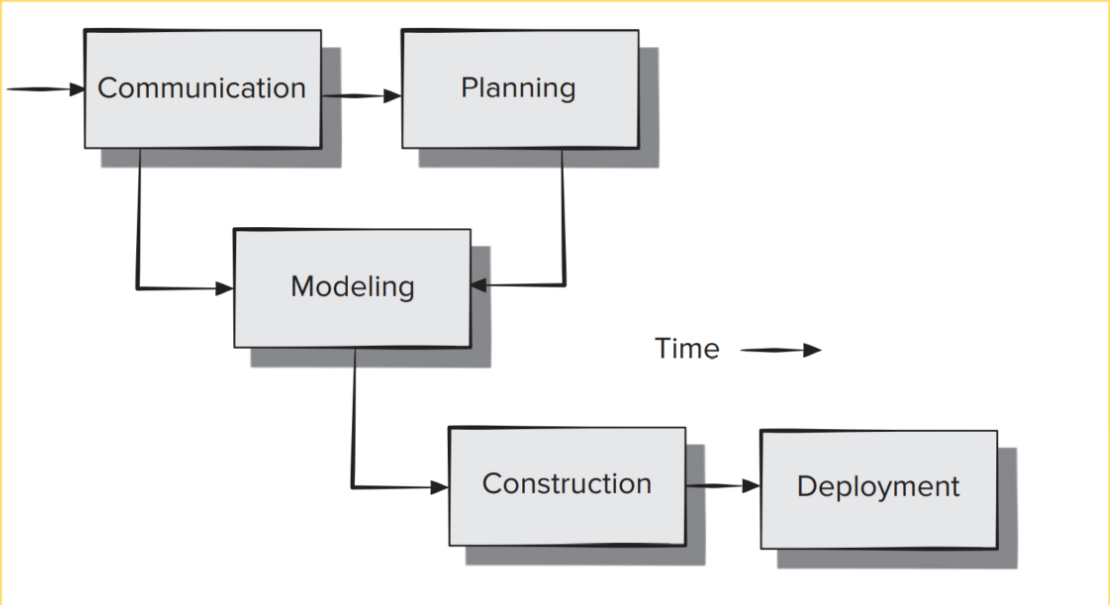


AGILE DEVELOPMENT: SCRUM

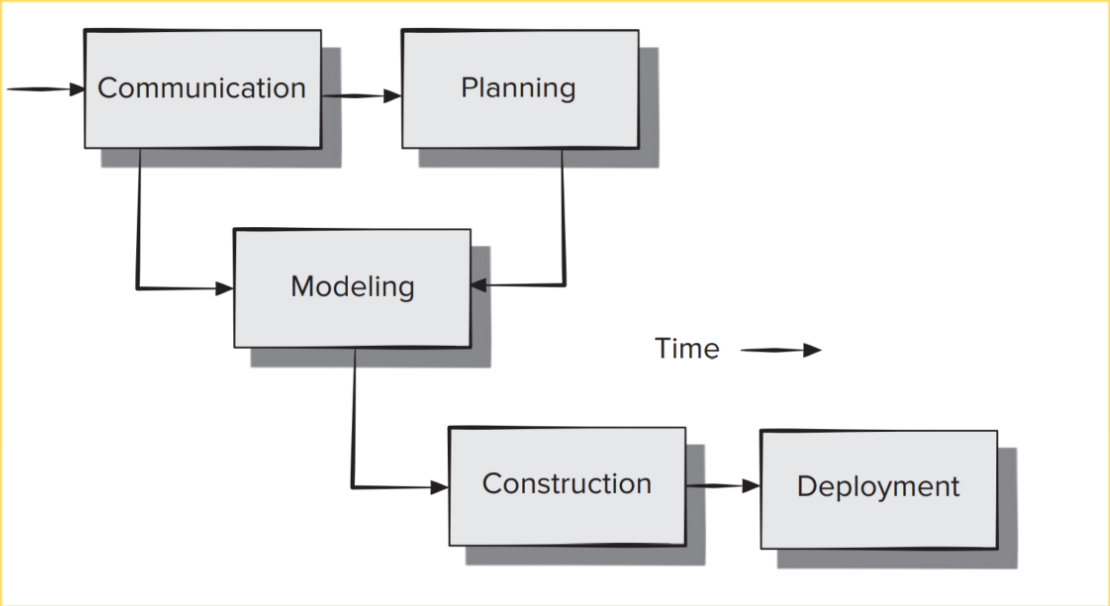


- **Daily Scrum Meeting (15 minutes, master & development, sometimes owner):**
  - **What did you do since the last team meeting?**
  - **What obstacles are you encountering?**
  - **What do you plan to accomplish by the next team meeting?**
- **Sprint Retrospective (3-hour for a 4-week sprint, master & deveopment)**
  - **What went well in the sprint**
  - **What could be improved**
  - **What the team will commit to improving in the next sprint**

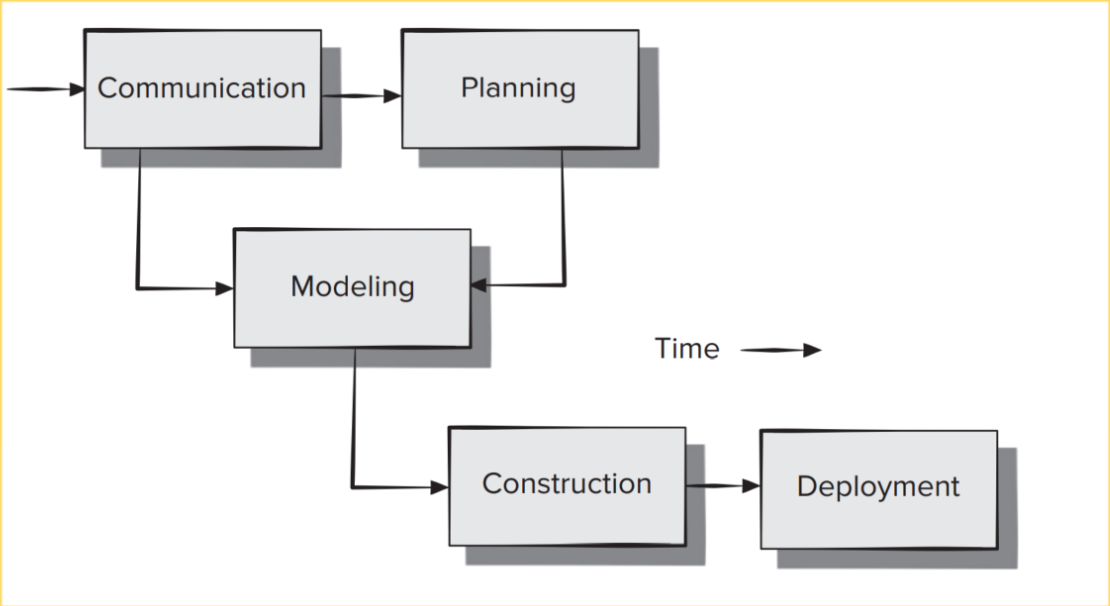
SOFTWARE PROCESS FRAMEWORK

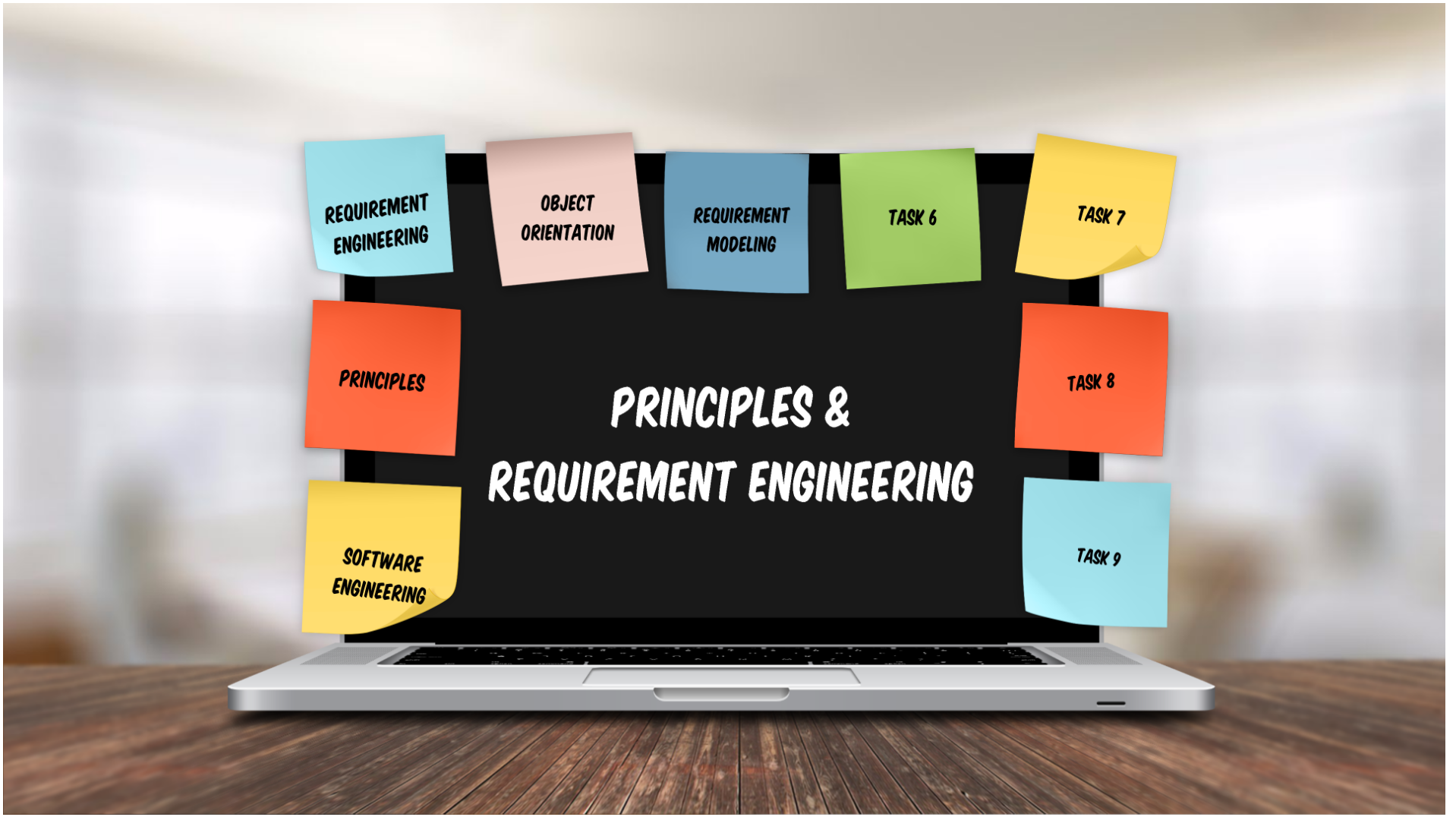


SOFTWARE PROCESS FRAMEWORK



SOFTWARE PROCESS FRAMEWORK




REQUIREMENT ENGINEERING
OBJECT ORIENTATION
REQUIREMENT MODELING
TASK 6
TASK 7
PRINCIPLES
PRINCIPLES &
REQUIREMENT ENGINEERING
TASK 8
SOFTWARE ENGINEERING
TASK 9

PRINCIPLES THAT GUIDE PRACTICE

- Separation of concerns
- Abstraction
- Consistency (e.g. placement of menu options, consistent color scheme, consistent use of recognizable icons)
- Transfer of information
  - errors, omissions, or ambiguity
  - analysis, design, construction, and testing of interfaces
- Modularity
  - interconnected simply
- Patterns
  - Architectural & design patterns
- Problem and its solution from several different perspectives (UML)
- Maintenance

PRINCIPLES THAT GUIDE PRACTICE

- Separation of concerns
- Abstraction
- Consistency (e.g. placement of menu options, consistent color scheme, consistent use of recognizable icons)
- Transfer of information
  - errors, omissions, or ambiguity
  - analysis, design, construction, and testing of interfaces
- Modularity
  - interconnected simply
- Patterns
  - Architectural & design patterns
- Problem and its solution from several different perspectives (UML)
- Maintenance

COMMUNICATION PRINCIPLES

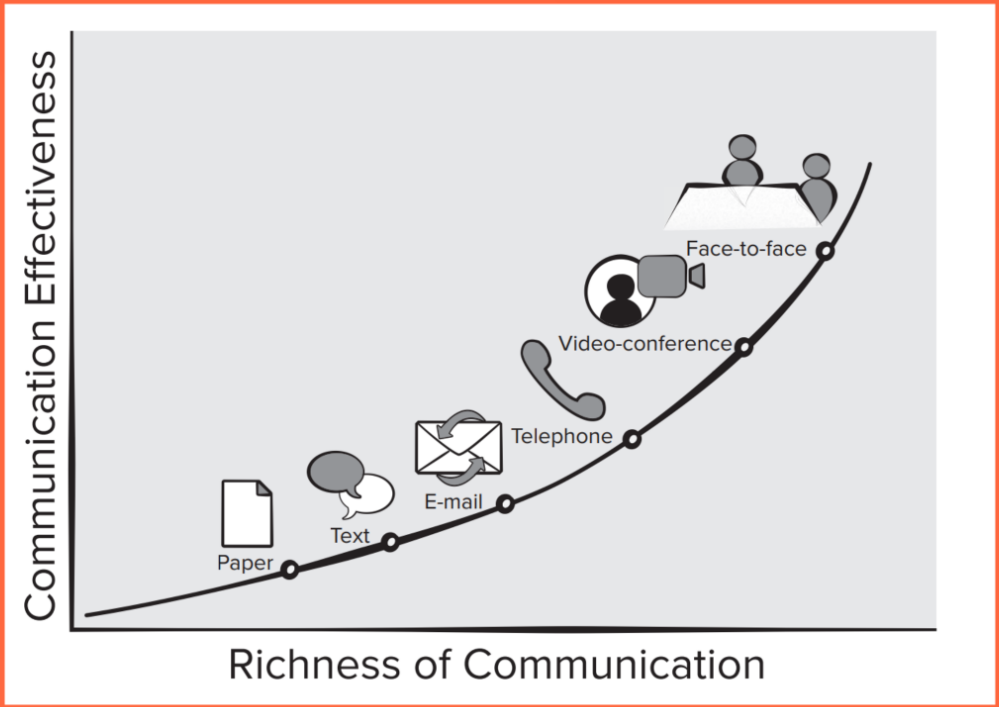
Customer

- Request for software
- Define overall business objectives
- Basic product requirements
- Coordinate funding

End User

- Use the software
- Define operational details

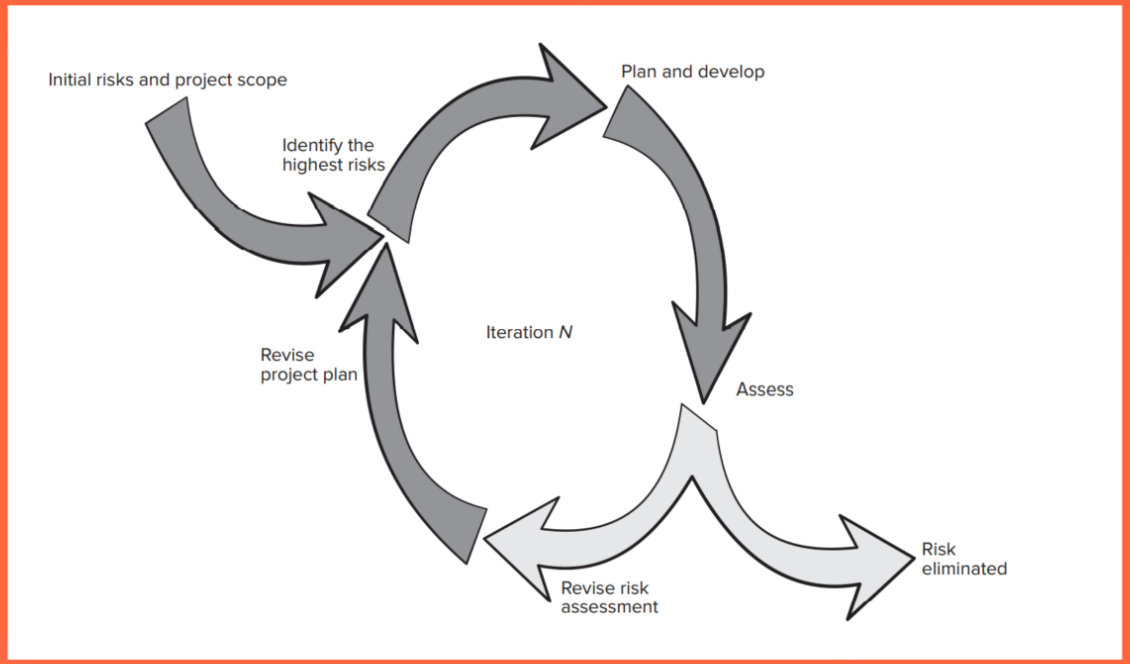
COMMUNICATION PRINCIPLES



COMMUNICATION PRINCIPLES

1. Know other's needs & listen patiently, clarify if needed
2. Prepare before communication: learn about problems, business domain jargon
   prepare agenda
3. Leader: keep the conversation productive, mediate conflict, follow principles
4. Face-to-face communication with material to focus
5. Take notes and document decisions
6. Use collective knowledge to describe product or system functions or features; build trust for a common goal;
7. Focus and modularize discussion; leave a topic after resolved
8. If unclear, draw a picture
9. agree/can't agree/unclear and cannot be clarified now, move on
10. Negotiate with compromise

PLANNING PRINCIPLES



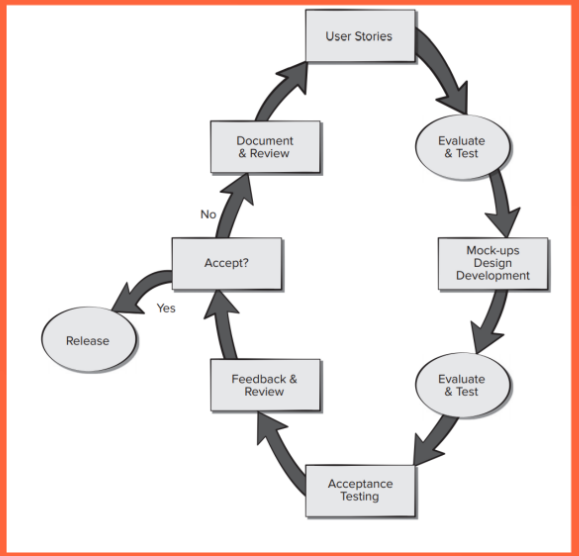
PLANNING PRINCIPLES

1. Scope: destination of the project
2. Stakeholders: priorities, project constraints
   negotiate: order of delivery, time lines.....
3. Plans will be revised on feedback
4. Contingency plan for risk with high impact and high probability
5. Consider realities: rest, changes, mistakes
6. Adjust granularity according to time available
7. How to ensure quality: technical review schedule, pair programming plan
8. Accommodate changes: change request submit, impact and cost assessment, implementation
9. Track (daily) & adjust as required

MODELING PRINCIPLES

1. Make models fast or none. Model for communication.
2. Create only models needed & kept up-to-date
3. Keep it simple, so software is easy to integrate/test/change/maintain
4. Amenable to change, but reasonable & complete
5. Models with explicit purpose or none
6. Fits with your product
7. First for communication, second for consistency. Syntax is not important.
8. Feedback as soon as possible
9. Track (daily) & adjust as required

CONSTRUCTION PRINCIPLES



Coding:

- Source code with programming language
- Automatic code generation: component representation
- 4th-generation programming language: Unreal4 Blueprints

Testing:

- Unit testing
- Integration testing (system)
- Functional testing (requirement)
- Structural testing (architecture/logics)
- Validation testing (requirement of complete system/software increment)
- Acceptance testing (all required features and functions, by customer)

CONSTRUCTION PRINCIPLES: CODING

Preparation before coding:

- Understand the problem to solve
- Basic design principles & concepts
- A programming language fitting functional & performance requirements, constraint
- Create unit tests

Coding:

- Algorithms by structured programming
- Consider pair programming
- Data structures fit design
- Interfaces consistent with software architecture

CONSTRUCTION PRINCIPLES: CODING

Validation after first coding pass:

- Code walkthrough
- Unit test and correct errors
- Refactor code to improve quality

CONSTRUCTION PRINCIPLES: TESTING

Testing principles:

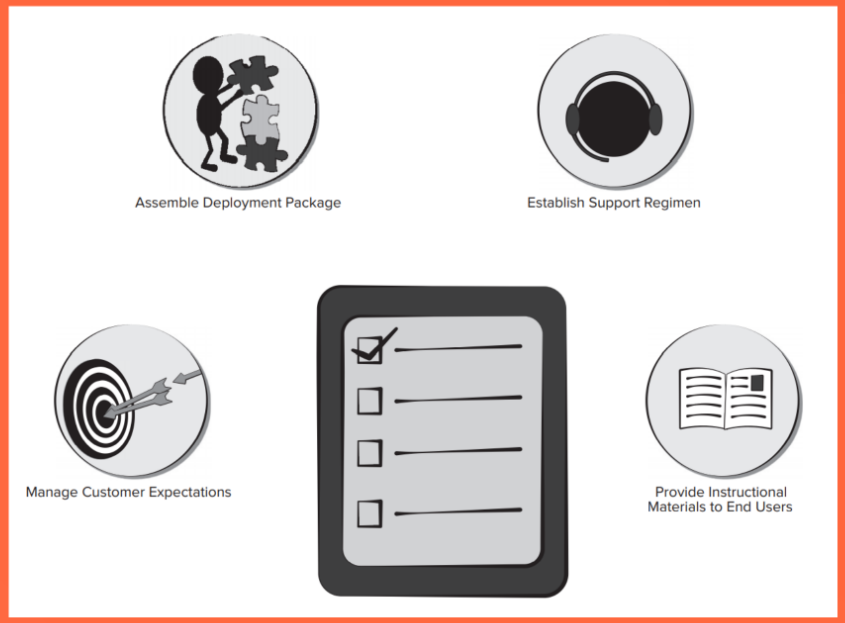
- Execute program to find error
- Good test case: high probability to find as-yet undiscovered error
- Tests traceable to customer requirements
- Tests planned early before use (e.g. when design model solidified)
- Pareto principle: 80% errors traceable to 20% program components (e.g. newest/least understood modules), so isolate suspect components & test thoroughly
- Cover program logic & exercise on all conditions in component-level design
- Test docs: requirements, specifications, code walk-throughs, and user manuals (85% defects)
- Track & look for patterns in defects
  - Software quality: total defects
  - Software stability: types of defects
  - Patterns of defects -> numbers of expected defects
- Include test cases(regression test cases) showing software is behaving correctly after change

DEPLOYMENT PRINCIPLES

Delivery

Support

Feedback



DEPLOYMENT PRINCIPLES

Delivery

Support

Feedback

1. Manage customer expectations: consistent to promise
2. Complete delivery package assembled & tested with actual user
   - Assembled & beta-tested: executable software, support data files, support documents
   - Installation scripts & operational features exercised in all configurations: hardware, operating systems, peripheral devices, networking
3. Support regimen before delivery: support plans & material, record-keeping mechanisms
4. Instructions to end users: training, troubleshooting guidelines, increments notes
5. Fix bug before delivery

REQUIREMENTS MODELING PRINCIPLES

Information

Function

Behavior

1. Information domain:
   - Data in: end users, other systems, external devices
   - Data out: via user interface, network interfaces, reports, graphics
   - Data stores: collect and organize persistent data objects
2. Function domain:
   - Transform data
   - Control: internal software processing, external system elements
   - Levels of abstraction vary
3. Behavior domain: consequence of external events
   - User input
   - Control data of external system
   - Monitoring data over a network
4. Contingency plan for risk with high impact and high probability
5. Consider realities: rest, changes, mistakes
6. Adjust granularity according to time available
7. How to ensure quality: technical review schedule, pair programming plan

DESIGN MODELING PRINCIPLES

Architecture

UI

Component

1.
2. Stakeholders: priorities, project constraints
   negotiate: order of delivery, time lines.....
3. Plans will be revised on feedback
4. Contingency plan for risk with high impact and high probability
5. Consider realities: rest, changes, mistakes
6. Adjust granularity according to time available
7. How to ensure quality: technical review schedule, pair programming plan
8. Accommodate changes: change request submit, impact and cost assessment, implementation
9. Track (daily) & adjust as required

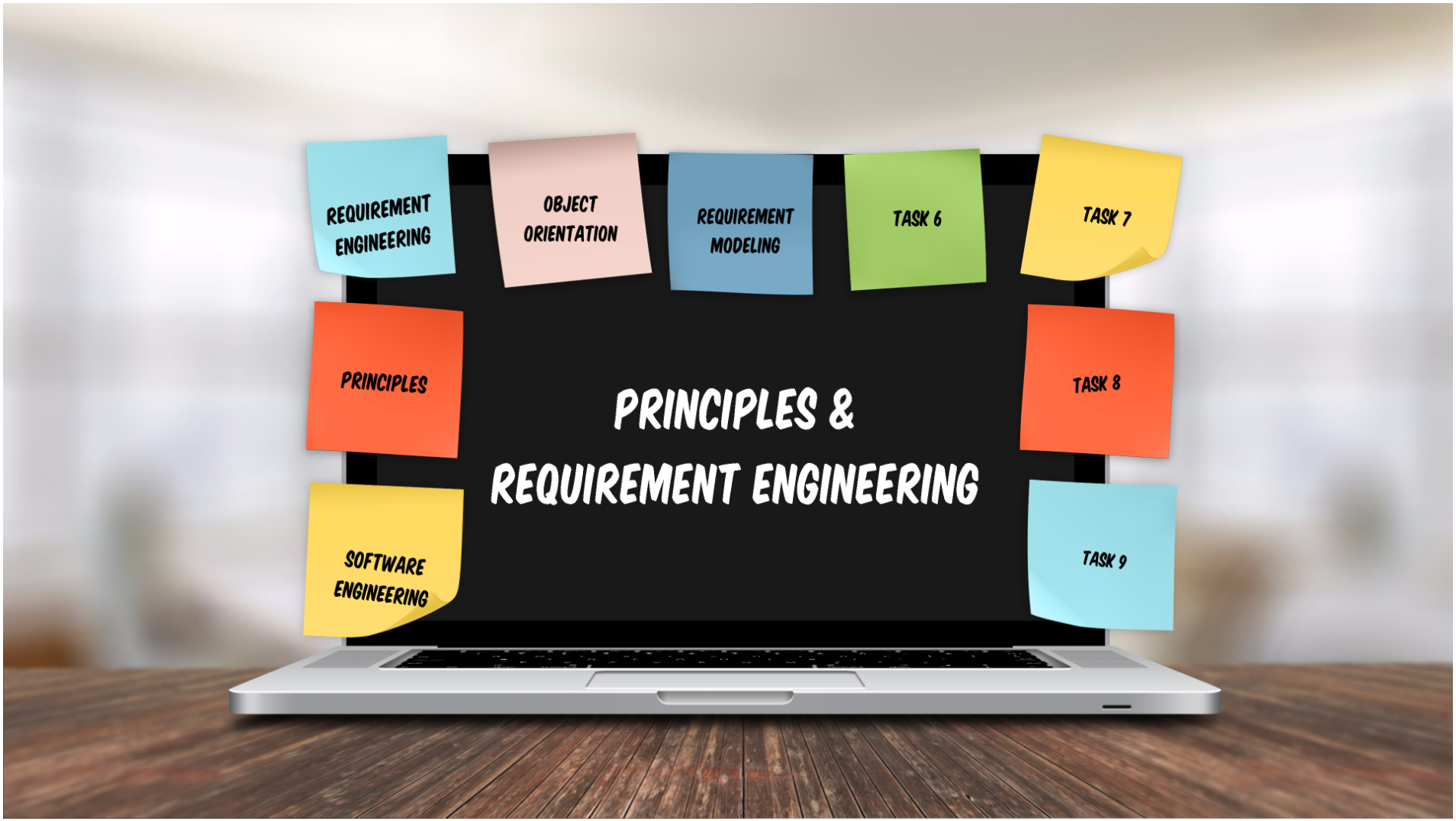
REQUIREMENTS MODELING PRINCIPLES

Information

Function

Behavior

1. Scope: destination of the project
2. Stakeholders: priorities, project constraints
   negotiate: order of delivery, time lines.....
3. Plans will be revised on feedback
4. Contingency plan for risk with high impact and high probability
5. Consider realities: rest, changes, mistakes
6. Adjust granularity according to time available
7. How to ensure quality: technical review schedule, pair programming plan
8. Accommodate changes: change request submit, impact and cost assessment, implementation
9. Track (daily) & adjust as required


REQUIREMENT ENGINEERING
OBJECT ORIENTATION
REQUIREMENT MODELING
TASK 6
TASK 7
PRINCIPLES
PRINCIPLES &
REQUIREMENT ENGINEERING
TASK 8
SOFTWARE ENGINEERING
TASK 9

REQUIREMENT ENGINEERING

1. Inception: basic problems, nature of solution, stakeholders, build communications
    - Ask stakeholder: who else should I talk to? what else should I ask? Are you the right one to answer?Is your answer official? What to solve? What is the business environment and a good solution?
    - Marketing: potential market, easy to sell
    - Business managers: within budget, fit defined market window
    - End users: features familiar & easy to learn & use
    - Software engineers: functions invisible to nontechnical stakeholders but enable infrastructure for more marketable functions & features
    - Support engineers: maintainability
    - Source for solution
2. Elicitation: business goals (long-term, functional/nonfunctional, precise), how to use daily
    - Base of latter steps
    - Define common/conflict/inconsistent requirements
    - Continuous changes
    - Work product: user scenarios (interaction with the system by end users)
3. Elaboration: refined requirements model
    - User scenarios -> analysis classes(attributes, services required, relationships & collaboration among classes)
    - Planning Poker

REQUIREMENT ENGINEERING

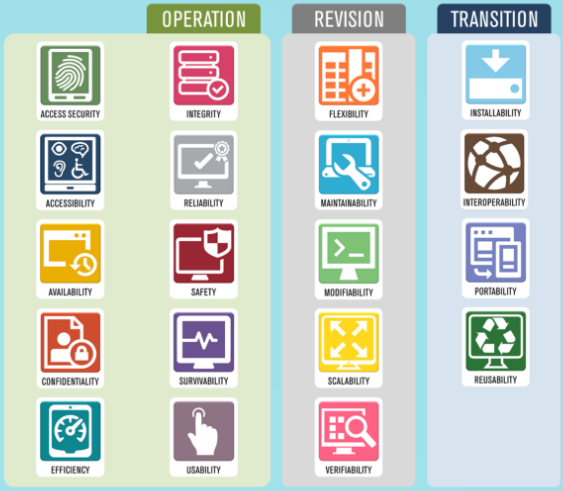
1. Inception: basic problems, nature of solution, stakeholders, build communications
    - Ask stakeholder: who else should I talk to? what else should I ask? Are you the right one to answer?Is your answer official? What to solve? What is the business environment and a good solution?
    - Marketing: potential market, easy to sell
    - Business managers: within budget, fit defined market window
    - End users: features familiar & easy to learn & use
    - Software engineers: functions invisible to nontechnical stakeholders but enable infrastructure for more marketable functions & features
    - Support engineers: maintainability
    - Source for solution
2. Elicitation: business goals (long-term, functional/nonfunctional, precise), how to use daily
    - Base of latter steps
    - Define common/conflict/inconsistent requirements
    - Continuous changes
    - Work product: user scenarios (interaction with the system by end users)
3. Elaboration: refined requirements model
    - User scenarios -> analysis classes(attributes, services required, relationships & collaboration among classes)
    - Planning Poker

REQUIREMENT ENGINEERING

4. Negotiation:rank requirements, discuss conflicts in priority, assesses cost & risk
    - iterative process for win-win
    - requirements eliminated, combined, and/or modified
5. Specification:written document, graphical models, formal mathematical model, usage scenarios, prototype
6. Validation (technical review): consistent, unambiguous, complete, correct, unrealistic
7. Requirements Management

Nonfunctional requirement list:

- Column: NFRs
- Row: software engineering guidelines
- Value: priority



Quality attribute	Requirement definition	Scope/How
Scalability	The number of concurrent players in the game is 2 – n.	Client-Server architecture style Component Game Server at server side
Portability	The game can be played with devices supporting either GPRS or UMTS connection. End-user devices support: PalmOS, EPOC, WinCE.	Client application. This requirement is realized by component Communication Manager Layered architecture style Client application includes a virtual machine layer
Extendibility	Features can be enriched, hence ensuring service evolution (in the WISE project, evolution is simulated by carrying out three development iterations).	Extension points in the architecture and code. - On the Server side e.g. quest sending, fights & attacks, manage high score list - On the Client side e.g. quest receiving and handling, fights/attacks/defending, buy/sell items
Modifiability	Services should be easily modified under the evolution of mobile terminals' hardware capabilities.	Client application. By separating communication manipulation and game management, and by splitting logically related functionality, modification is easier. Of course any modification is isolated if interfaces are not influenced.

REQUIREMENT ENGINEERING

7. Requirements Management
   - Clear: can it be misinterpreted?
   - Source (e.g., a person, a regulation, a document) of the requirement identified? Final requirement be examined by the source.
   - Quantitative bound of terms
   - Dependency/related requirements: cross-reference matrix
   - system
   - Violation of domain constraints?
   - Testable: validation criteria
   - Traceable to any system model, overall system and product objectives?
   - Easy understanding/reference/translation into more technical work products
   - Index for the specification
   - Implicit requirements
   - Requirements associated with performance, behavior, and operational characteristics

Internal requirements <--> Implementation				
	IntReq. 1	IntReq. 2	IntReq. 3	IntReq. 4
Impl. 1	x			
Impl. 2				
Impl. 3			x	
Impl. 4				x
Impl. 5				

Implementation <--> Test case			
	TestCase. 1	TestCase. 2	TestCase. 3
Impl. 1	x		
Impl. 2		x	
Impl. 3			x
Impl. 4			
Impl. 5			

Customer requirements <--> Internal requirements				
	IntReq. 1	IntReq. 2	IntReq. 3	IntReq. 4
CustReq. 1	x		x	
CustReq. 2		x		
CustReq. 3				

Customer requirements <--> Test case			
	TestCase. 1	TestCase. 2	TestCase. 3
CustReq. 1	x		x
CustReq. 2			
CustReq. 3			

SHOWCASE

1. SafeHome Product Request by a marketing person:
   - Market for home management systems growing @ 40%/year
   - First SafeHome function: home security
   - “Alarm system”: familiar, easy to sell
   - Voice control (Alexa)
   - Against and/or recognize: illegal entry, fire, flooding, carbon monoxide levels……
   - Wireless sensors, programmable to homeowner
   - Automatic contact to monitoring agency, owner’s cell phone

2. After reading, participants should list:
   - Objects part of the environment that surrounds the system
   - Objects to be produced by the system
   - Objects used by the system to perform its functions
   - Services (processes or functions) manipulate/interact with the objects
   - Constraints: cost, size, business rules
   - Performance criteria: speed, accuracy, security

SHOWCASE

3. List by participants:
   - Object: control panel, smoke detectors, window & door sensors, motion detectors, an alarm, an event (a sensor has been activated), a display, a tablet, telephone numbers, a telephone call
   - List of services: configuring the system, setting the alarm, monitoring the sensors, dialing the phone using a wireless router, programming the control panel, and reading the display (note that services act on objects).
   - Constraints: must recognize when sensors are not operating, must be user friendly, must interface directly to a standard phone line
   - Performance criteria: a sensor event should be recognized within 1 second, an event priority scheme should be implemented

4. Consensus list:
   - - redundant entries
   - + new ideas during discussion

SHOWCASE

5. Use case: user story (text), template, graph
    - Actor: people/device communicates with product in function/behavior defined (external)
    - End user-multiple roles VS Actor-single role in a use case
    - SafeHome showcase:
        - Placement mode - editor
        - Testing mode - tester
        - Monitoring mode - monitor
        - Troubleshooting mode - troubleshooter
    - Primary actors (direct & frequent): use system function, get benefit from the system
    - Secondary actors: support the system so that primary actors can do their work

    - Actors: primary, secondary & their goals
    - Preconditions before the story
    - Actors' tasks/ functions
    - Exceptions of stories (e.g. forget password while login)
    - Variations in the actor’s interaction
    - Information acquired, produced, changed, desired by the actor
    - Will the actor inform the system about external changes?

SHOWCASE

SafeHome actors:

- Homeowner (a user)
- Setup manager (the same person as homeowner, but in a different role)
- Sensors (devices attached to the system)
- Monitoring and response subsystem (the central station that monitors the SafeHome home security function)

Interaction with SafeHome by homeowner:

- Alarm control panel
- Tablet
- Cell phone

Functions for homeowner:

1. Enters a password to allow all other interactions
2. Inquires about the status of a security zone
3. Inquires about the status of a sensor
4. Presses the panic button in an emergency
5. Activates/deactivates the security system

SHOWCASE

System activation use case by homeowner:

1. Ready to input?
   - No: not ready message -> close window/doors -> not ready message disappears
2. keypad: input four-digit password -> ==valid password?
   - No: beep & reset panel for additional input
   - Yes: wait for further action
3. Activates different sensors by selection:
   - Stay: only activate perimeter sensors
   - Away: activate all sensors
4. Activation success: red alarm light

Use case:	*InitiateMonitoring*
Primary actor:	Homeowner.
Goal in context:	To set the system to monitor sensors when the homeowner leaves the house or remains inside.
Preconditions:	System has been programmed for a password and to recognize various sensors.
Trigger:	The homeowner decides to "set" the system, that is, to turn on the alarm functions.

Scenario:

1. Homeowner observes control panel.
2. Homeowner enters password.
3. Homeowner selects "stay" or "away."
4. Homeowner observes red alarm light to indicate that *SafeHome* has been armed.

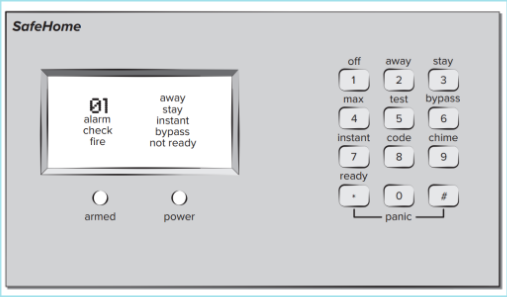
Exceptions:

1. Control panel is *not ready:* Homeowner checks all sensors to determine which are open and then closes them.
2. Password is incorrect (control panel beeps once): Homeowner reenters correct password.
3. Password not recognized: Monitoring and response subsystem must be contacted to reprogram password.
4. *Stay* is selected: Control panel beeps twice, and a *stay* light is lit; perimeter sensors are activated.
5. *Away* is selected: Control panel beeps three times, and an *away* light is lit; all sensors are activated.

Priority:	Essential, must be implemented
When available:	First increment
Frequency of use:	Many times per day
Channel to actor:	Via control panel interface
Secondary actors:	Support technician, sensors
Channels to secondary actors:	
	Support technician: phone line
	Sensors: hardwired and radio frequency interfaces

Open issues:

1. Should there be a way to activate the system without the use of a password or with an abbreviated password?
2. Should the control panel display additional text messages?
3. How much time does the homeowner have to enter the password from the time the first key is pressed?
4. Is there a way to deactivate the system before it actually activates?



SHOWCASE

System activation use case by homeowner:

1. Ready to input?
   - No: not ready message -> close window/doors -> not ready message disappears
2. keypad: input four-digit password -> ==valid password?
   - No: beep & reset panel for additional input
   - Yes: wait for further action
3. Activates different sensors by selection:
   - Stay: only activate perimeter sensors
   - Away: activate all sensors
4. Activation success: red alarm light

Use case:	*InitiateMonitoring*
Primary actor:	Homeowner.
Goal in context:	To set the system to monitor sensors when the homeowner leaves the house or remains inside.
Preconditions:	System has been programmed for a password and to recognize various sensors.
Trigger:	The homeowner decides to "set" the system, that is, to turn on the alarm functions.

Scenario:

1. Homeowner observes control panel.
2. Homeowner enters password.
3. Homeowner selects "stay" or "away."
4. Homeowner observes red alarm light to indicate that *SafeHome* has been armed.

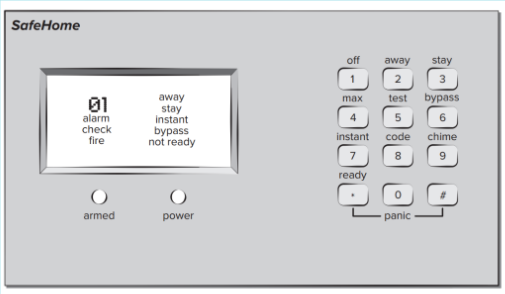
Exceptions:

1. Control panel is *not ready:* Homeowner checks all sensors to determine which are open and then closes them.
2. Password is incorrect (control panel beeps once): Homeowner reenters correct password.
3. Password not recognized: Monitoring and response subsystem must be contacted to reprogram password.
4. *Stay* is selected: Control panel beeps twice, and a *stay* light is lit; perimeter sensors are activated.
5. *Away* is selected: Control panel beeps three times, and an *away* light is lit; all sensors are activated.

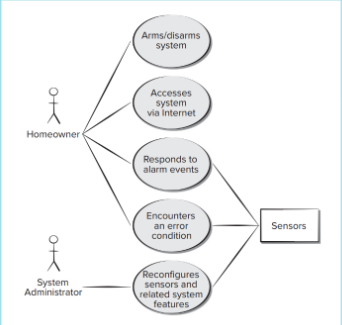
Priority:	Essential, must be implemented
When available:	First increment
Frequency of use:	Many times per day
Channel to actor:	Via control panel interface
Secondary actors:	Support technician, sensors
Channels to secondary actors:	
	Support technician: phone line
	Sensors: hardwired and radio frequency interfaces

Open issues:

1. Should there be a way to activate the system without the use of a password or with an abbreviated password?
2. Should the control panel display additional text messages?
3. How much time does the homeowner have to enter the password from the time the first key is pressed?
4. Is there a way to deactivate the system before it actually activates?



UML Use Case

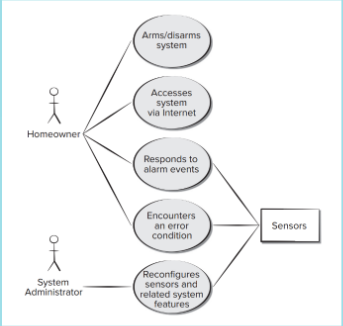


ANALYSIS MODEL

- Scenario-Based Elements: use case diagrams
- Class-Based Elements
  - Class: similar attributes & common behaviors
- Behavioral Elements
  - State: externally observable mode
  - Event: external stimuli cause transitions between states

ANALYSIS MODEL

UML Use Case



- Scenario-Based Elements: use case diagrams
- Class-Based Elements
  - Class: similar attributes & common behaviors
- Behavioral Elements
  - State: externally observable mode
  - Event: external stimuli cause transitions between states

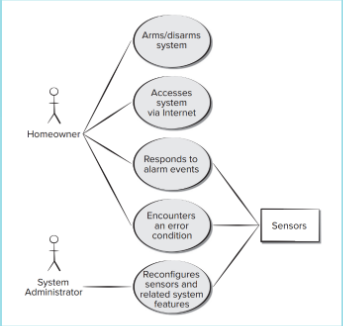
ANALYSIS MODEL

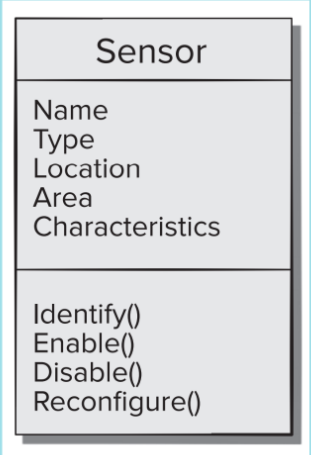
- Scenario-Based Elements: use case diagrams

- Class-Based Elements
  - Class: similar attributes & common behaviors

- Behavioral Elements
  - State: externally observable mode
  - Event: external stimuli cause transitions between states

UML Use Case





UML Class Diagram

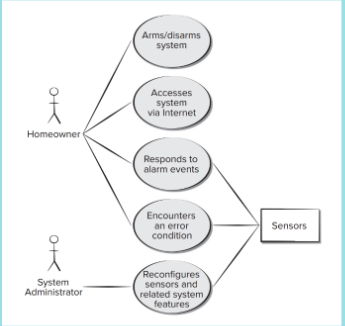
ANALYSIS MODEL

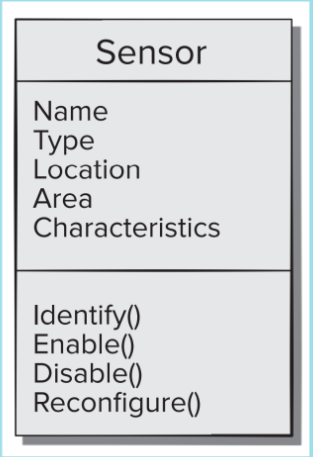
- Scenario-Based Elements: use case diagrams

- Class-Based Elements
  - Class: similar attributes & common behaviors

- Behavioral Elements
  - State: externally observable mode
  - Event: external stimuli cause transitions between states

UML Use Case





UML Class Diagram

UML State Diagram

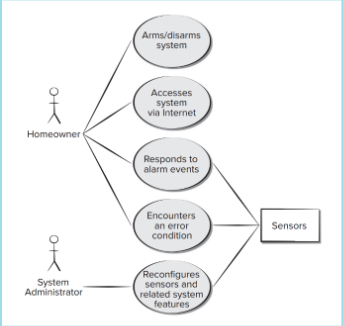
ANALYSIS MODEL

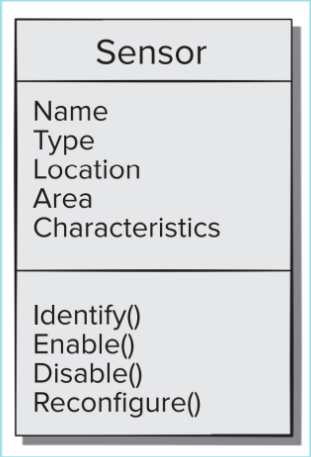
- Scenario-Based Elements: use case diagrams

- Class-Based Elements
  - Class: similar attributes & common behaviors

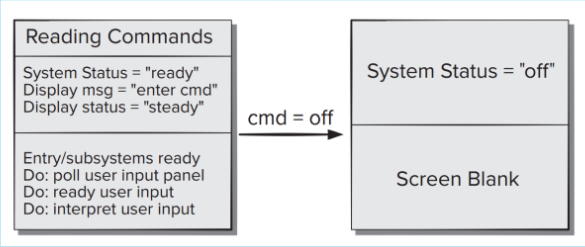
- Behavioral Elements
  - State: externally observable mode
  - Event: external stimuli cause transitions between states

UML Use Case





UML Class Diagram

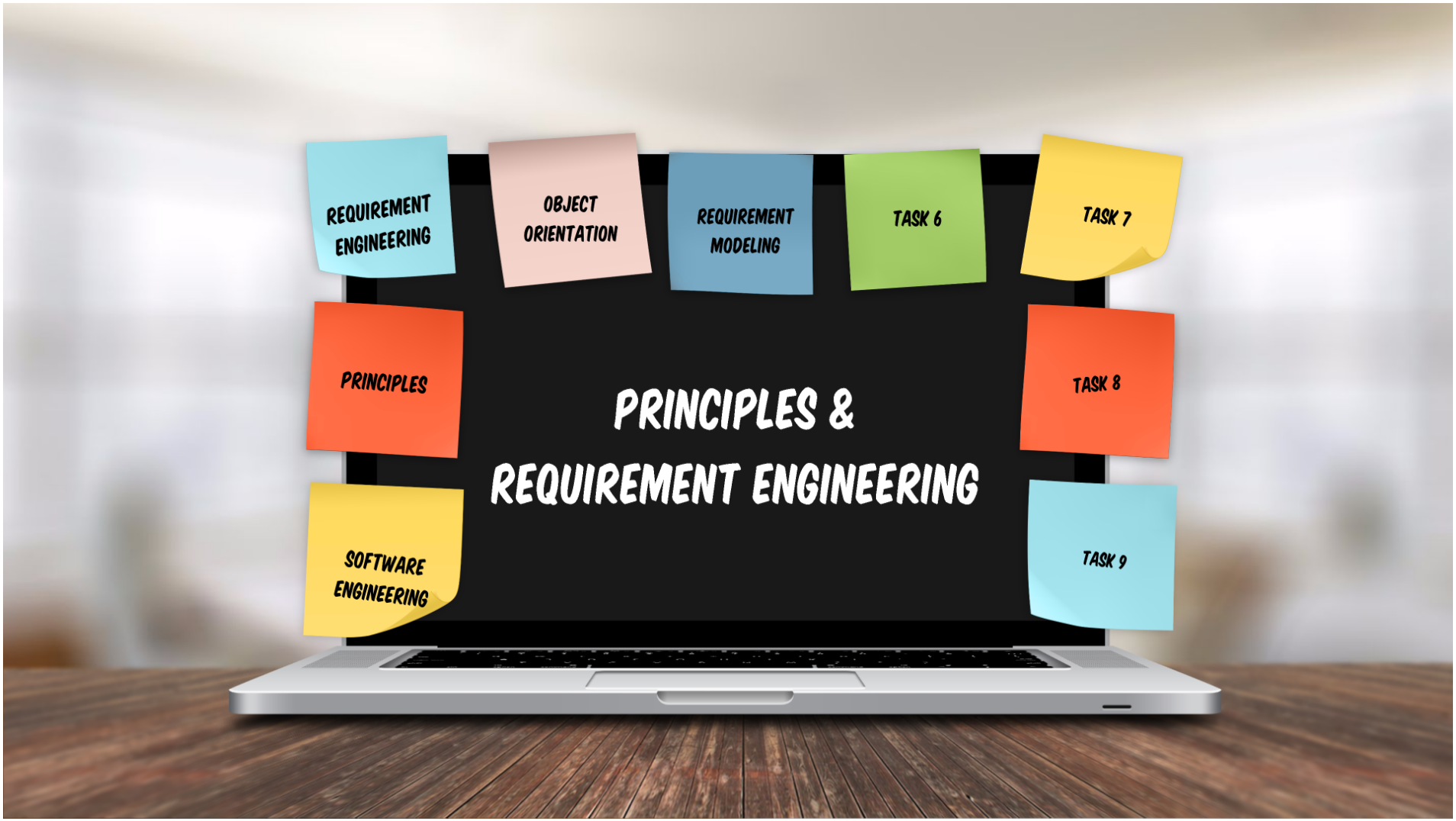


UML State Diagram

REQUIREMENT REVIEW

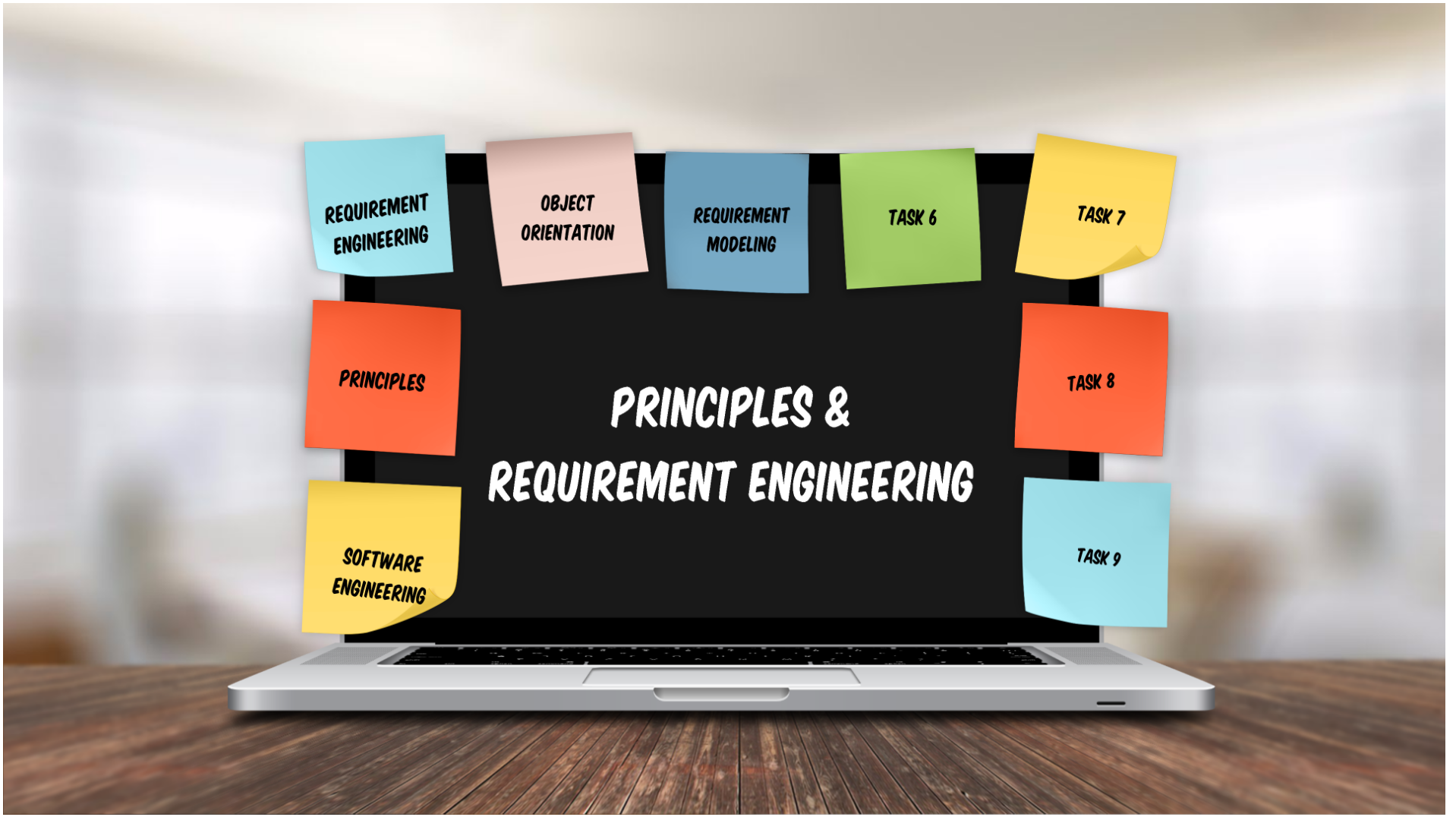
1. Each requirement consistent with the overall objectives for the system/product?
2. All requirements specified at the proper level of abstraction?Any requirements provide a level of technical detail that is inappropriate at this stage?
3. Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
4. Is each requirement bounded and unambiguous?
5. Is a source (generally, a specific individual) noted for each requirement?
6. Do any requirements conflict with other requirements?
7. Is each requirement achievable in the technical environment that will house the system or product?
8. Is each requirement testable, once implemented?
9. Does the requirements model properly reflect the information, function, and behavior of the system to be built?
10. Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system?
11. Have requirements patterns been used to simplify the requirements model? Have all patterns been properly validated? Are all patterns consistent with customer requirements?

Click to edit text


REQUIREMENT ENGINEERING
OBJECT ORIENTATION
REQUIREMENT MODELING
TASK 6
TASK 7
PRINCIPLES
PRINCIPLES &
REQUIREMENT ENGINEERING
TASK 8
SOFTWARE ENGINEERING
TASK 9

TASK 4

TASK 4

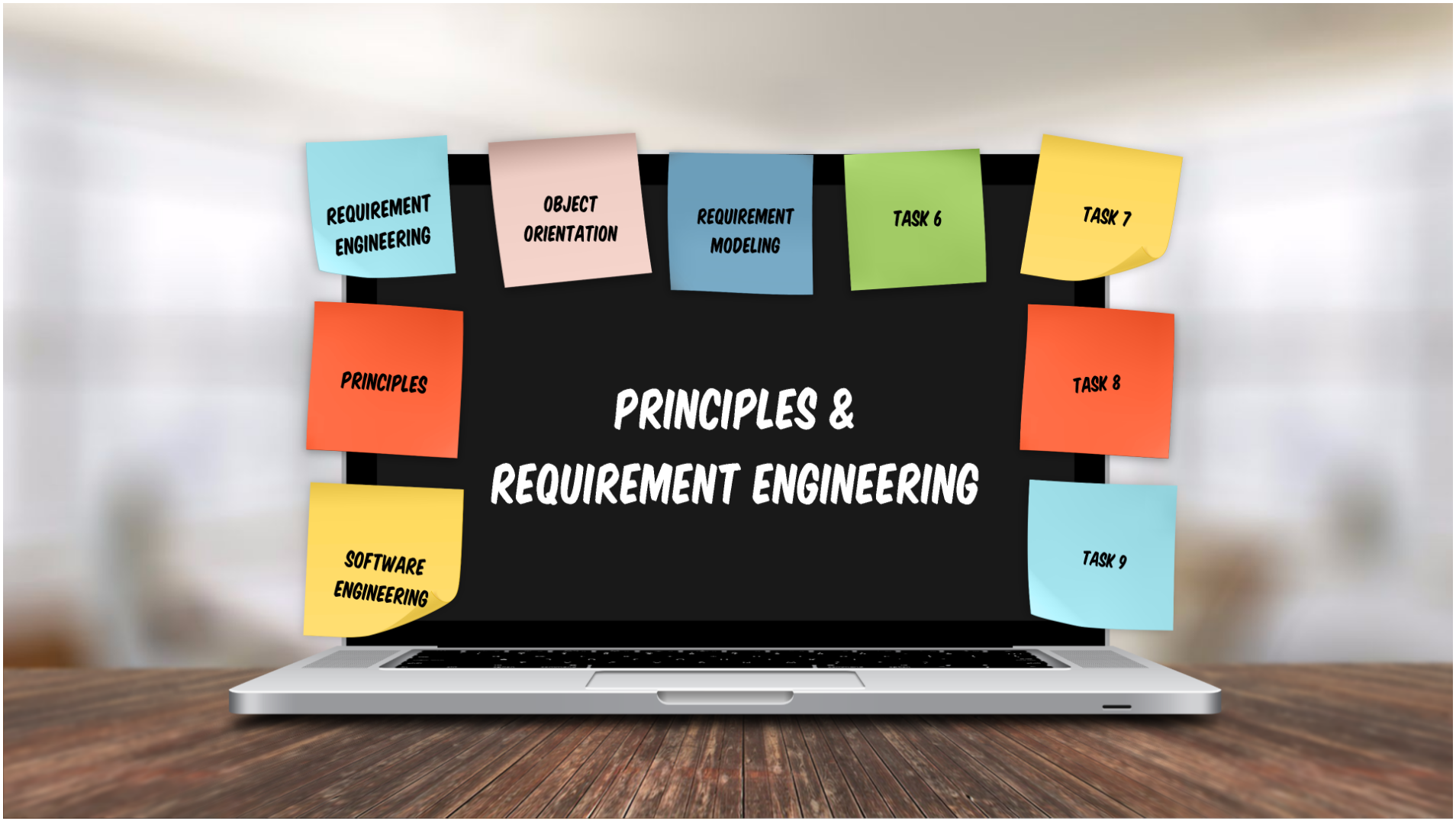

REQUIREMENT ENGINEERING
OBJECT ORIENTATION
REQUIREMENT MODELING
TASK 6
TASK 7
PRINCIPLES
PRINCIPLES &
REQUIREMENT ENGINEERING
TASK 8
SOFTWARE ENGINEERING
TASK 9

REQUIREMENT MODELING

- Scenario-based model: from the point of system “actors”
- Class-oriented model: object-oriented classes (attributes & operations), how classes collaborate to achieve system requirements
- Behavioral model: how the software reacts to internal/external “events”
- Data model: information domain for the problem
- · Flow-oriented models that represent the functional elements of the system and
- how they transform data as they move through the system.

REQUIREMENT MODELING

- Scenario-based model: from the point of system “actors”
- Class-oriented model: object-oriented classes (attributes & operations), how classes collaborate to achieve system requirements
- Behavioral model: how the software reacts to internal/external “events”
- Data model: information domain for the problem
- · Flow-oriented models that represent the functional elements of the system and
- how they transform data as they move through the system.

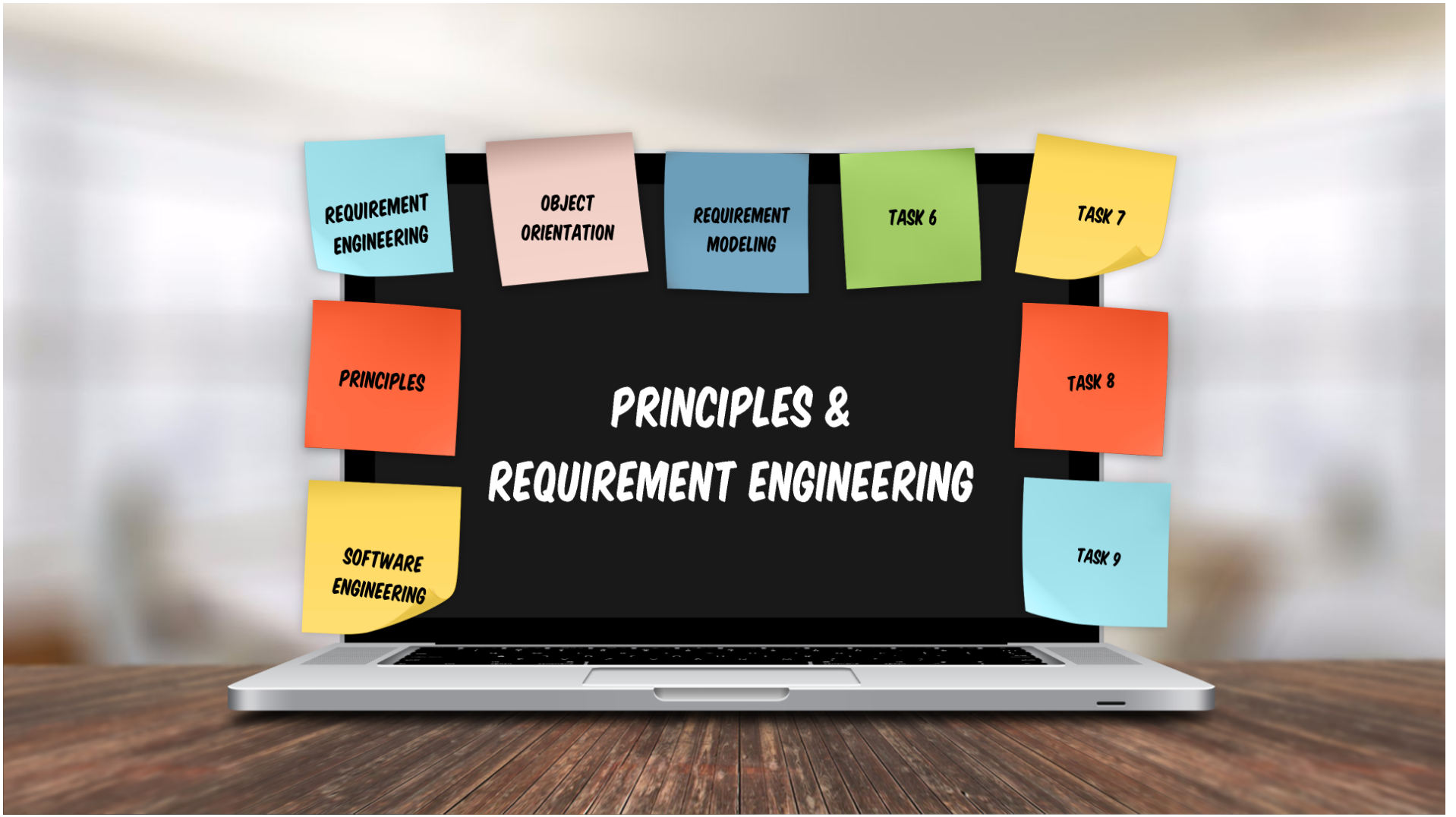

REQUIREMENT ENGINEERING
OBJECT ORIENTATION
REQUIREMENT MODELING
TASK 6
TASK 7
PRINCIPLES
PRINCIPLES &
REQUIREMENT ENGINEERING
TASK 8
SOFTWARE ENGINEERING
TASK 9

TASK 6

Click to edit text

TASK 6

Click to edit text


REQUIREMENT ENGINEERING
OBJECT ORIENTATION
REQUIREMENT MODELING
TASK 6
TASK 7
PRINCIPLES
PRINCIPLES &
REQUIREMENT ENGINEERING
TASK 8
SOFTWARE ENGINEERING
TASK 9