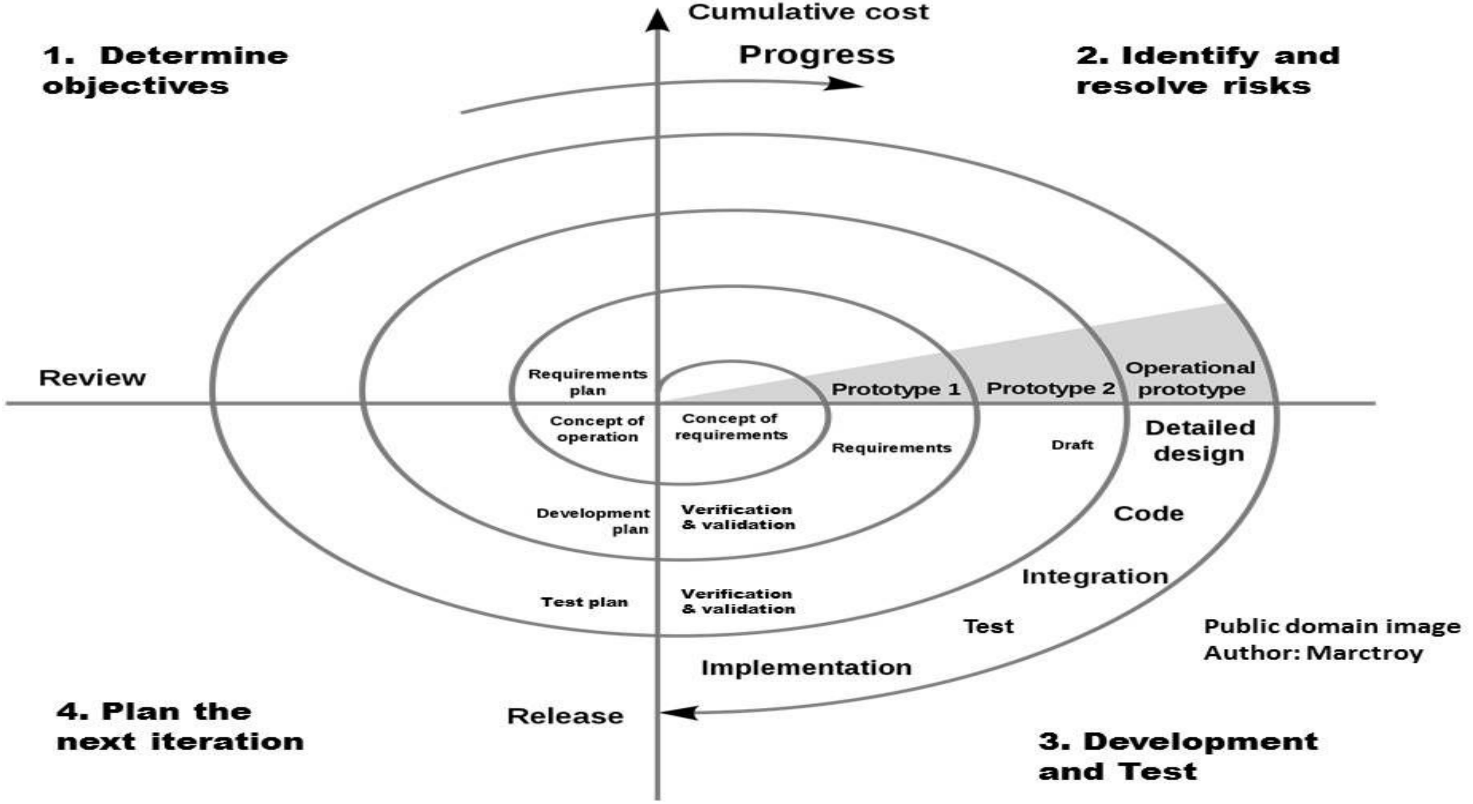
Software Engineering
Lecture 5: Design

Gregory S. DeLozier, Ph.D.

gdelozie@kent.edu

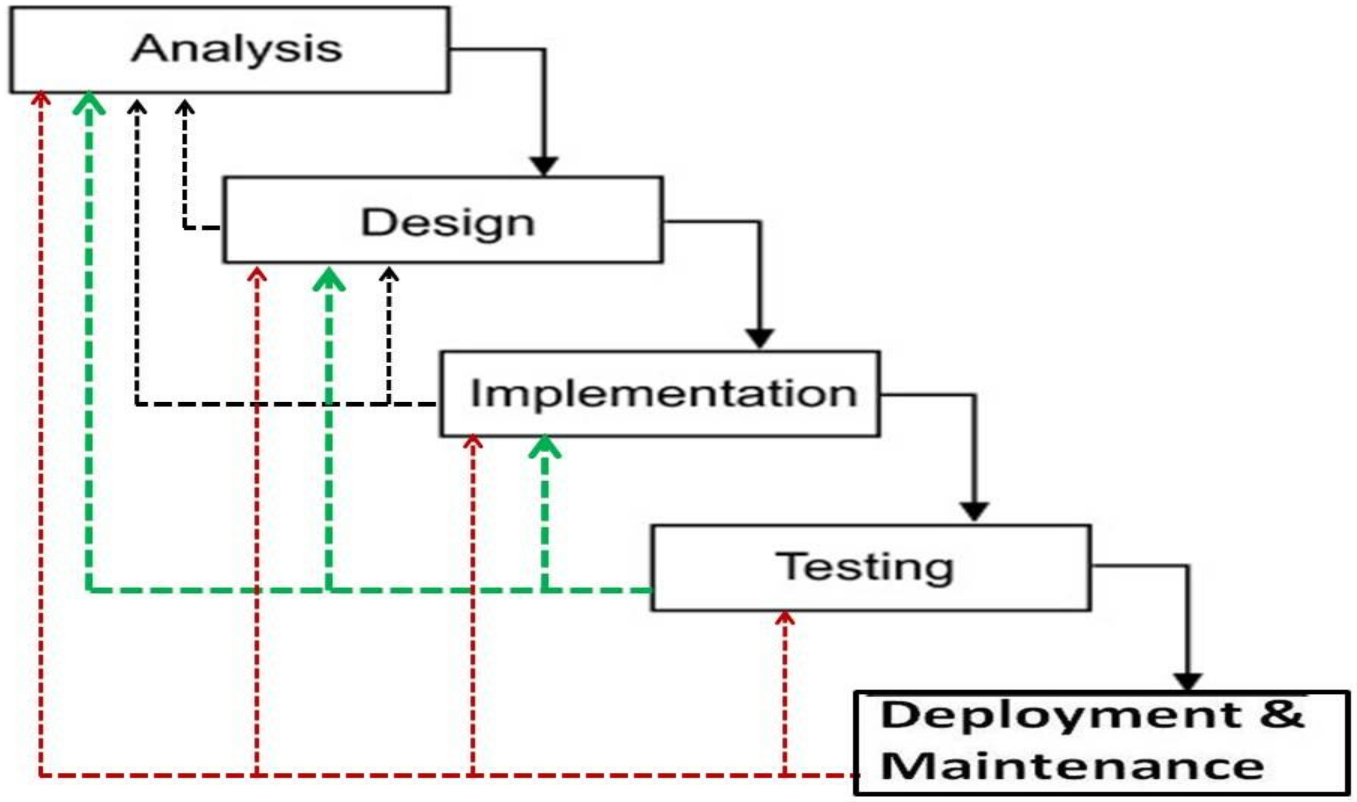
First, a little more about requirements

Boehm Spiral Model

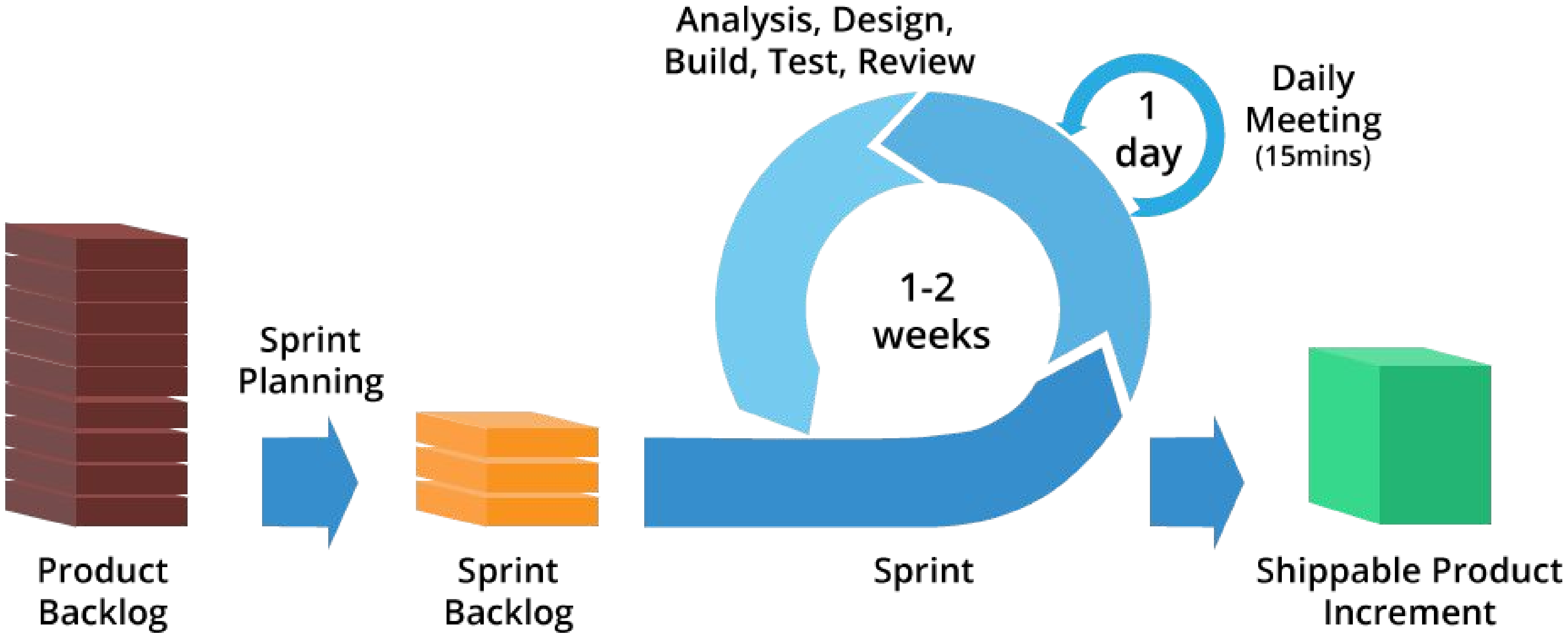


Public domain image
Author: Marctroy

Iterative Waterfall



Agile Software Development



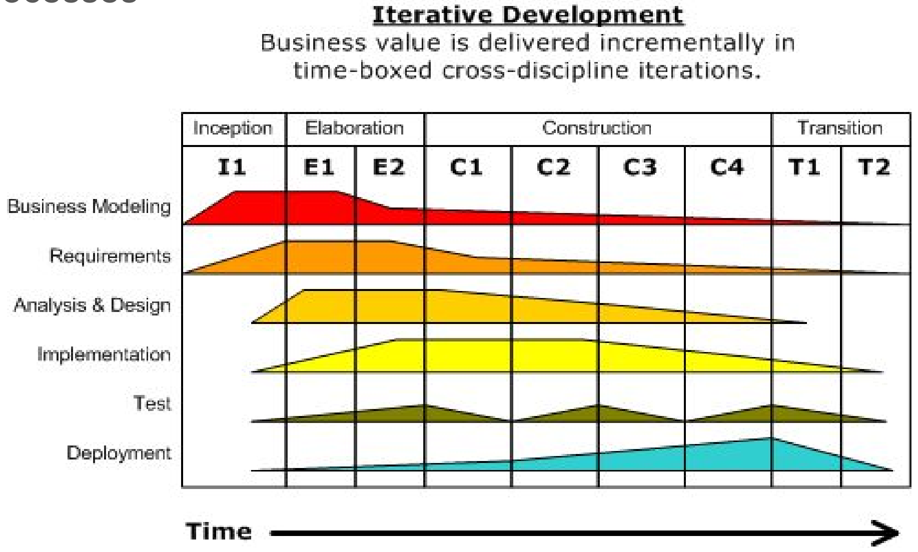
What's the Difference?

Purpose of Spiral & Iterative Waterfall

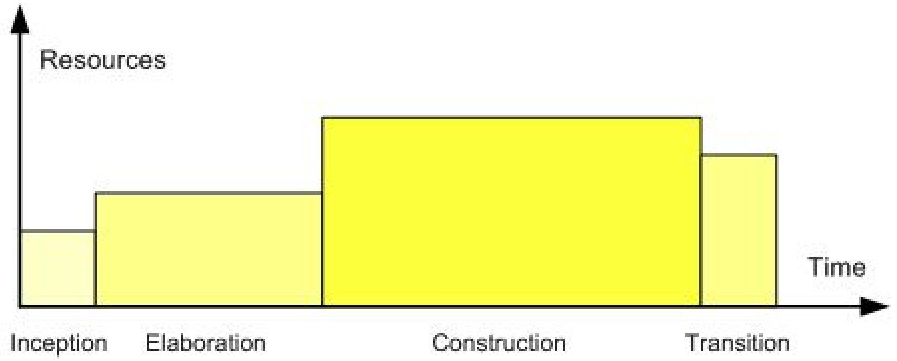
- In the spiral model
  - Each spiral knows more about the problem
  - Each spiral creates a better prototype
  - At the middle of the spiral, knowledge is perfect
  - At the middle of the spiral, perfect requirements -> perfect software
- The goal of iteration in waterfall
  - Correction of requirements found to be in error
  - Pursuit of precision in requirements
- This leads to…
  - Extensive requirements processes (we're packing for interplanetary travel…)
  - Extensive tooling to enforce extraordinary discipline

Tools for Traditional Development

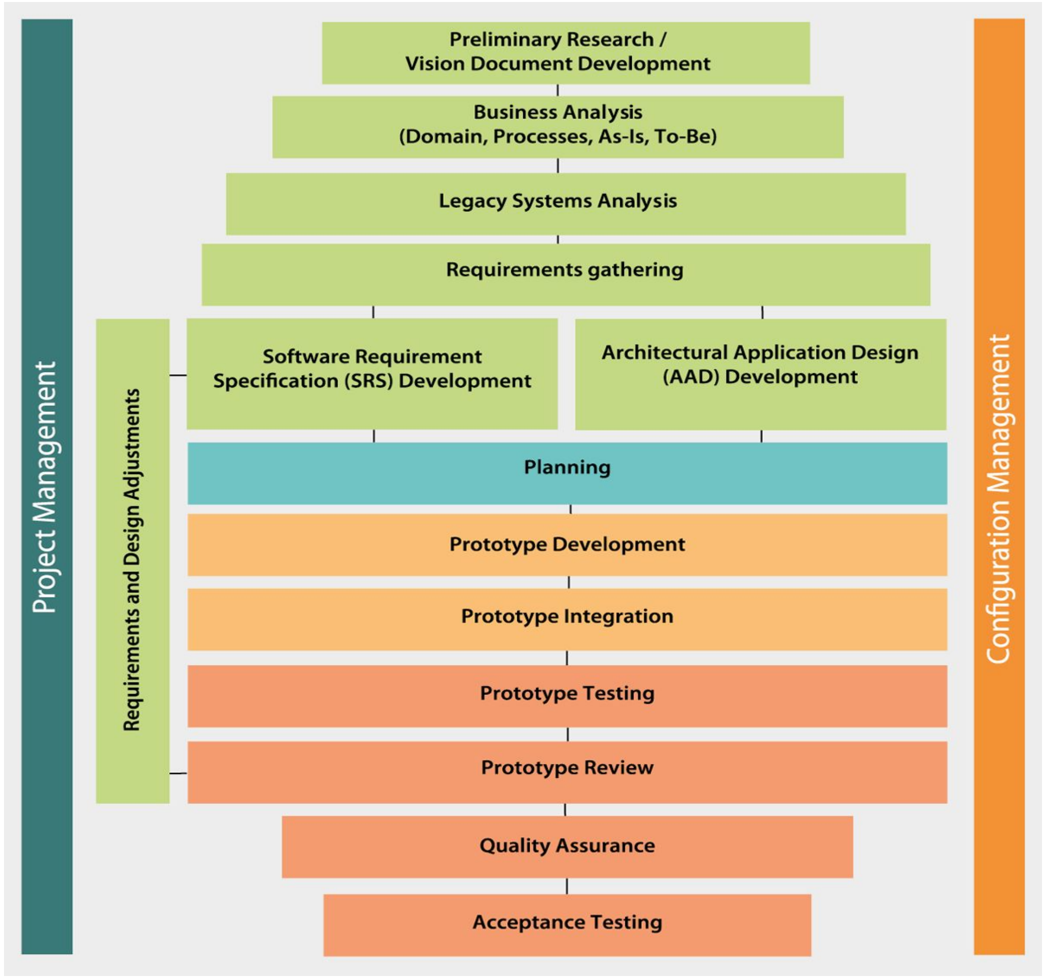
- Rational Unified Process
  - Sold by IBM
  - Prescribes dozens of very specific processes
  - Claims (now) to be iterative
  - Very BDUF
    - (big design up front)
    - notice requirements work
  - Sold by IBM
  - Very expensive
  - Check out the poster



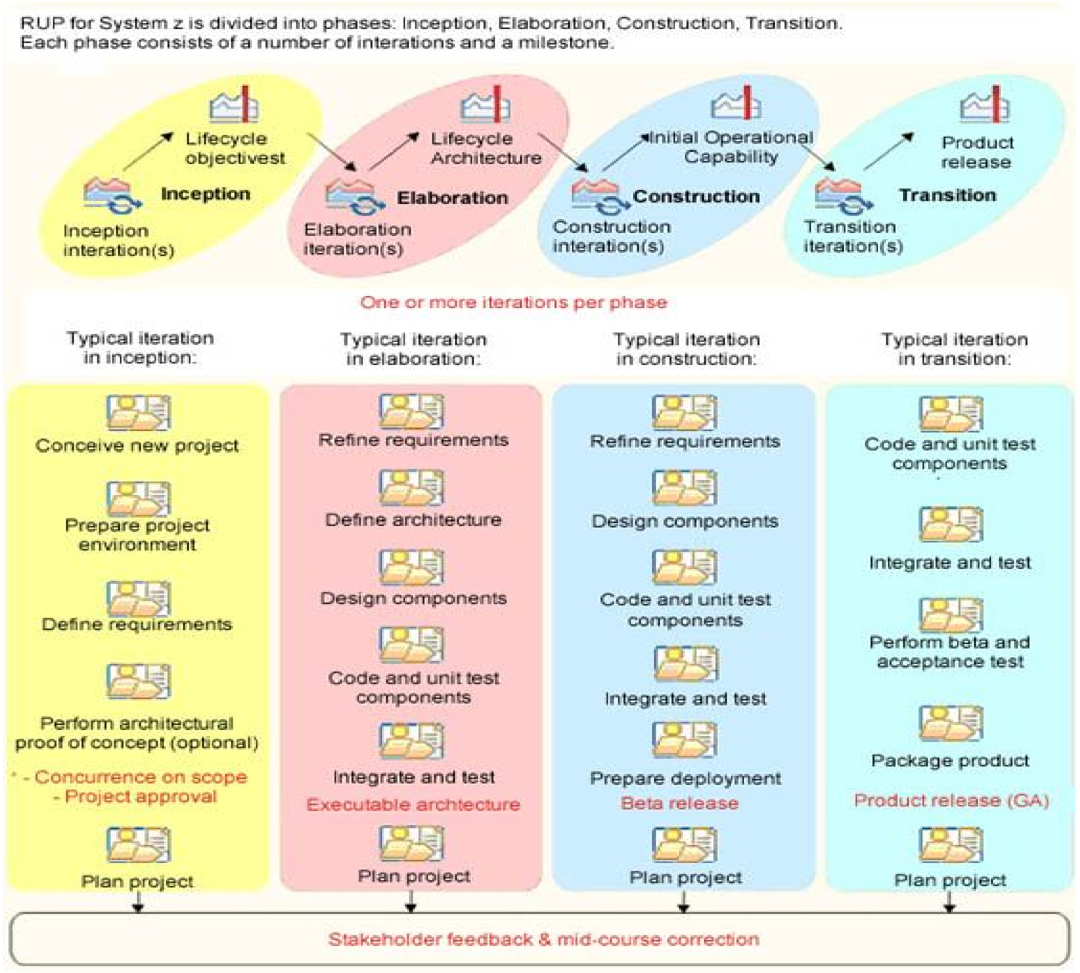
Rational...


Resources
Time
Inception
Elaboration
Construction
Transition

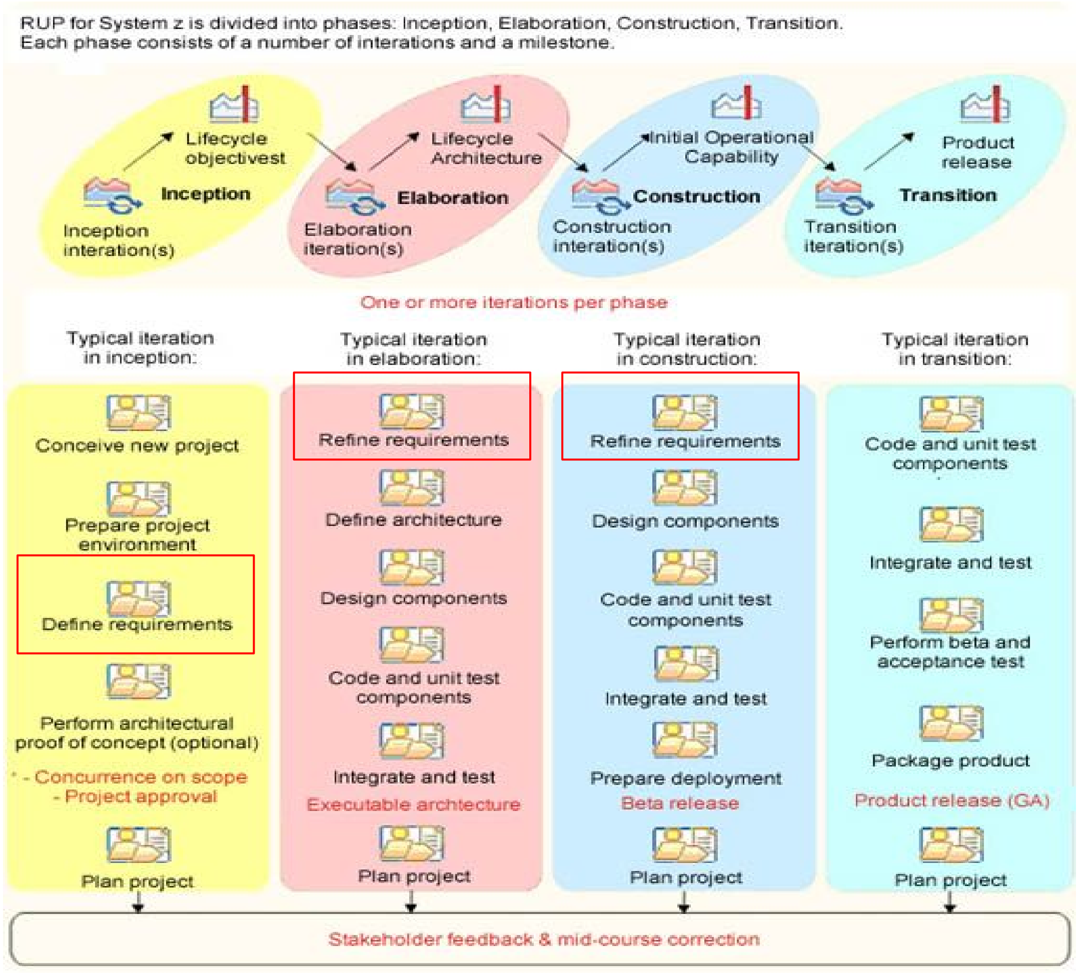
Rational...


Project Management
Preliminary Research / Vision Document Development
Business Analysis (Domain, Processes, As-Is, To-Be)
Legacy Systems Analysis
Requirements gathering
Requirements and Design Adjustments
Software Requirement Specification (SRS) Development
Architectural Application Design (AAD) Development
Planning
Prototype Development
Prototype Integration
Prototype Testing
Prototype Review
Quality Assurance
Acceptance Testing
Configuration Management

Rational...


RUP for System z is divided into phases: Inception, Elaboration, Construction, Transition.
Each phase consists of a number of interations and a milestone.
Lifecycle objectivest
Inception
Inception interation(s)
Lifecycle Architecture
Elaboration
Elaboration iteration(s)
Initial Operational Capability
Construction
Construction interation(s)
Product release
Transition
Transition iteration(s)
One or more iterations per phase
Typical iteration in inception:
Conceive new project
Prepare project environment
Define requirements
Perform architectural proof of concept (optional)
* - Concurrence on scope
- Project approval
Plan project
Typical iteration in elaboration:
Refine requirements
Define architecture
Design components
Code and unit test components
Integrate and test
Executable archtecture
Plan project
Typical iteration in construction:
Refine requirements
Design components
Code and unit test components
Integrate and test
Prepare deployment
Beta release
Plan project
Typical iteration in transition:
Code and unit test components
Integrate and test
Perform beta and acceptance test
Package product
Product release (GA)
Plan project
Stakeholder feedback & mid-course correction

Rational...


RUP for System z is divided into phases: Inception, Elaboration, Construction, Transition.
Each phase consists of a number of interations and a milestone.
Lifecycle objectivest
Inception
Inception interation(s)
Lifecycle Architecture
Elaboration
Elaboration iteration(s)
Initial Operational Capability
Construction
Construction interation(s)
Product release
Transition
Transition iteration(s)
One or more iterations per phase
Typical iteration in inception:
Conceive new project
Prepare project environment
Define requirements
Perform architectural proof of concept (optional)
- Concurrence on scope
- Project approval
Plan project
Typical iteration in elaboration:
Refine requirements
Define architecture
Design components
Code and unit test components
Integrate and test
Executable archtecture
Plan project
Typical iteration in construction:
Refine requirements
Design components
Code and unit test components
Integrate and test
Prepare deployment
Beta release
Plan project
Typical iteration in transition:
Code and unit test components
Integrate and test
Perform beta and acceptance test
Package product
Product release (GA)
Plan project
Stakeholder feedback & mid-course correction

Rational...

Where this leads: there are literally hundreds of different work products, activities, and considerations called out on this poster of the entire RUP workflow.

The sticky notes are optional. My copy doesn't have them. :-)

Where does this leave us?

- There is a lot invested in this model
- People that know it or have learned it
- People that sell it
    - (both vendors and practitioners)
- The minority of situations that really need it
    - (Going to Mars, maybe?)
    - (Robotic brain surgery, maybe?)
- This leads to a question:
    - "What is so great about agile that isn't true about *our* existing iterative processes?"
    - ...and it's a legitimate question.

And what's the answer?

- Old theory (iterative waterfall, Boehm, etc)
  - Problems (unpredictability, scheduling problems, useless software) occur…
  - Due to incomplete understanding of what must be done
  - Due to imprecise and incomplete work in defining requirements (or specifications)
  - so…
  - To improve software and the predictability of results…
  - ***...we must take the time to improve the requirements and specification.***

- New theory (Agile)
  - We will *never* ***(without a time machine)*** have a complete, perfect set of requirements
  - Nevertheless we have to write software, since the alternative is not to deliver value at all
  - We must create a way to write software in the presence of errors
  - ***If we can tolerate errors, endlessly eliminating them at great expense is a waste of time, and puts the delivery of any value at all at risk.***

The last word on the matter...

- Back when this was a really hot topic of discussion
- Ken Schwaber wrote a great note.

    http://jeffsutherland.org/scrum/Schwaber_on_CMM.html

- It's so important I'm going to cover it in class.
- Read it again, anyway.
- In case it ever disappears, it's on the next slide.

Ken Schwaber's Note about Agile

From: Ken Schwaber [ken.schwaber@verizon.net]
Sent: Sunday, September 15, 2002 4:53 PM
To: Jeff Sutherland
Subject: RE: Waterfall methodology
Jeff,
Reading Shawn Presson's comments regarding CMM, waterfall, and agile reminded me of Barry Boehm's comments at XP/Agile Universe'02 and the comments of several CMM authors at INCOSE'02. I was reminded of the good intentions and sound footings of CMM and Waterfall, both initiated to address problems in systems development and systemic project failures rates that have only marginally improved over the decades.

I was vastly amused by Presson's suspicion that we're only contrasting agile and previous approaches to sell books. If I could even afford to drink Starbuck's coffee with book royalties, I would be pleased. The contrast is painted from two perspectives: the current state of CMM and waterfall, and the theoretical contrast between CMM/waterfall and agile processes.

I was challenged at INCOSE'02; "how do you intend to prevent agile from being turned into a pile of junk, as has happened to CMM." CMM was transformed from its principles and vision into its practices, KPA's, certification, and institutionalization by DoD implementations, consultants, carpetbaggers and everyone else trying to make a buck. The same thing happened with structured methods, information engineering, and CASE tools that would generate code from design and design from code. All reduced to overhead and rubble. Of course the kernels of modeling, precision of expression, thinking before coding, and having a process persist, but for many the weight of the commercial implementations have done their damage.
So, some of the contrast and reaction of "agilists" to CMM and waterfall are to current implementations, not the vision nor the intentions. All of you have to is look at CMM as implemented at most major organizations or waterfall as implemented in such methodologies as Navigator to understand the damage. I was brought into one CMM Level 3 shop because it was so badly implemented that development had literarally stopped. Problems couldn't be reduced to an adequate level of agreement and definition to proceed with coding. However, the rest of the contracst occurs because CMM, waterfall spring from a different theoretical basis than agile processes. Increased precision and definition are at their core, which is one of the alternatives for controlling a process. However, this approach only works when the definition and the problem have a mapping approaching 1:1 and the problem domain (technical, requirements, and people) is relatively simple: "It is typical to adopt the defined (theoretical) modeling approach when the underlying mechanisms by which a process operates are reasonably well understood. When the process is too complicated for the defined approach, the empirical approach is the appropriate choice." *Process Dynamics, Modeling, and Control*, Ogunnaike and Ray, Oxford University Press, 1992.

The agile process is based on the empirical approach, accepting the complexity of the problem and addressing it through frequent inspection and constant adaptation. Empiricism is seen even in the Agile Manifesto, where the signatories pose agility as an empirical counterpoint to a defined approach, rather than as a new set of defined absolutes. The various agile processes implement empiricism with the underlying practices of iterative development, frequent inspection of status and increments of work, self-organizing teams, emerging architecture and requirements, and solid collaboration. You will not find many detailed task definitions, pert charts, or assignments in agile processes; by contrast, CMM and waterfall progressively implement themselves by increasing the amount of detail.

Agile is not a silver bullet. It is simply a correct approach to complex process control. It reflects a lean manufacturing approach to software development, and – as manufacturing started doing twenty years ago – eschews the Taylor model of defined process control.

I expect the success of agile processes to be challenged by the same degenerative, buck-making processes that hurt CMM and waterfall. I expect adopting agile processes to be quite difficult in many organizations because of the degree of change required; indeed, our agile processes have primarily succeeded when organizations absolutely need the projects to succeed and can't risk failure. I expect agile to also be challenged by the abysmal state of engineering practices in our industry, such as code management, release control, testing, and builds. Agile implementations highlight poor engineering practices immediately, whereas model and definition-based approaches let this failure slide until near the end of the project.

Shawn Presson is a kindred spirit, not a foe. We all are trying to improve the software development process.

Ken Schwaber

Design

Software Design

- The process of implementing solutions to problems using software
- This is where we describe *how* we will solve the problem
- Based on our understanding of software capabilities
- Based on our understanding of the requirements
- Based on our understanding of constraints

- We tend to imagine solutions to problems
- Conveying (and sharing) our visions leads to communication problems
- Tools are useful here

Design Principles - Davis

- Davis - 201 Software Principles
    - *https://books.google.com/books/about/201_Principles_of_Software_Development.html?id=8HAhAQAAIAAJ*
- These are quoted in many places
    - *https://en.wikipedia.org/wiki/Software_design*

Design Principles

The design process should not suffer from "tunnel vision." A good designer should consider alternative approaches, judging each based on the requirements of the problem, the resources available to do the job.

Design Principles

The design should be traceable to the analysis model. Because a single element of the design model often traces to multiple requirements, it is necessary to have a means for tracking how requirements have been satisfied by the design model.

Design Principles

The design should not reinvent the wheel. Systems are constructed using a set of design patterns, many of which have likely been encountered before. These patterns should always be chosen as an alternative to reinvention. Time is short and resources are limited! Design time should be invested in representing truly new ideas and integrating those patterns that already exist.

Design Principles

The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world. That is, the structure of the software design should (whenever possible) mimic the structure of the problem domain.

Design Principles

The design should exhibit uniformity and integration. A design is uniform if it appears that one person developed the entire thing. Rules of style and format should be defined for a design team before design work begins. A design is integrated if care is taken in defining interfaces between design components.

Design Principles

The design should be structured to accommodate change. The design concepts discussed in the next section enable a design to achieve this principle.

Design Principles

The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered. Well- designed software should never "bomb": it should be designed to accommodate unusual circumstances, and if it must terminate processing, do so in a graceful manner.

Design Principles

Design is not coding, coding is not design.

Even when detailed procedural designs are created for program components, the level of abstraction of the design model is higher than source code. The only design decisions made at the coding level address the small implementation details that enable the procedural design to be coded.

Design Principles

The design should be assessed for quality as it is being created, not after the fact. A variety of design concepts and design measures are available to assist the designer in assessing quality.

Design Principles

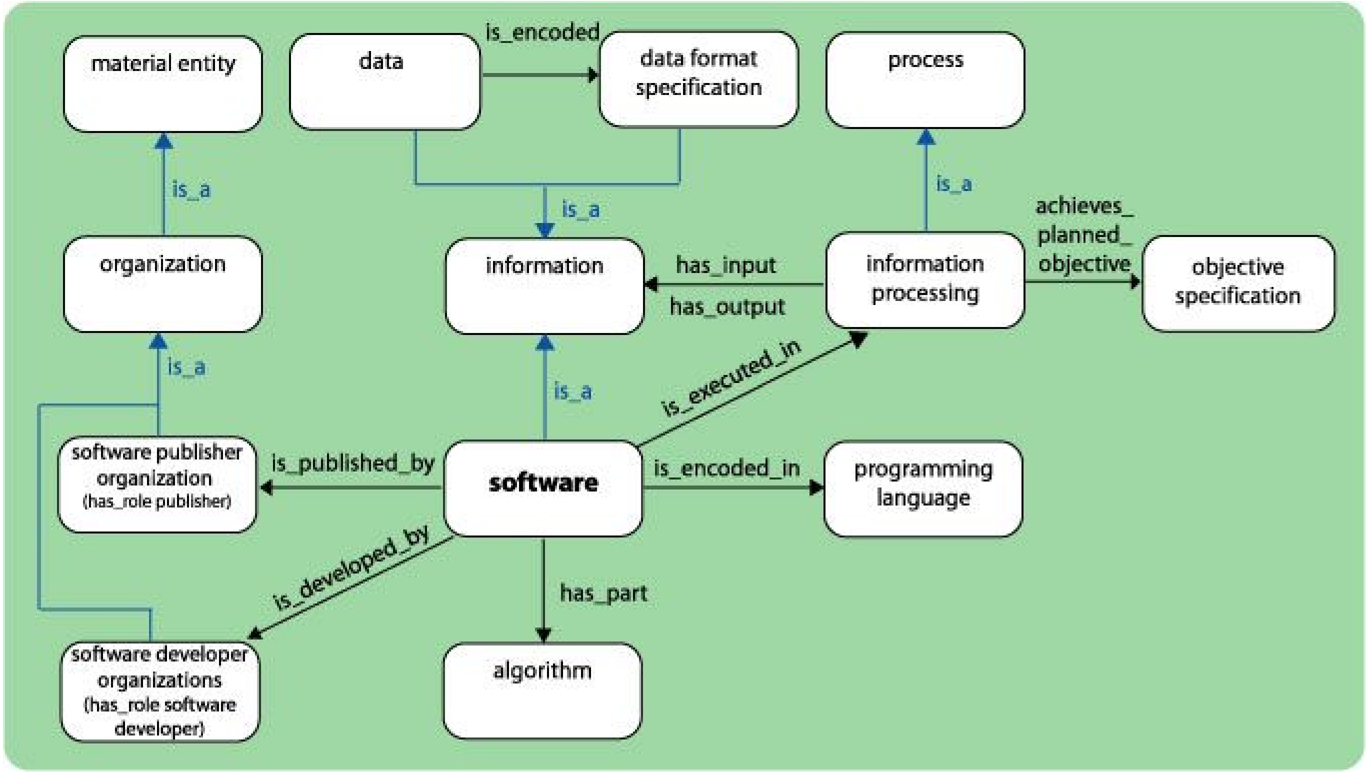
The design should be reviewed to minimize conceptual (semantic) errors. There is sometimes a tendency to focus on minutiae when the design is reviewed, missing the forest for the trees. A design team should ensure that major conceptual elements of the design (omissions, ambiguity, inconsistency) have been addressed before worrying about the syntax of the design model.

Models

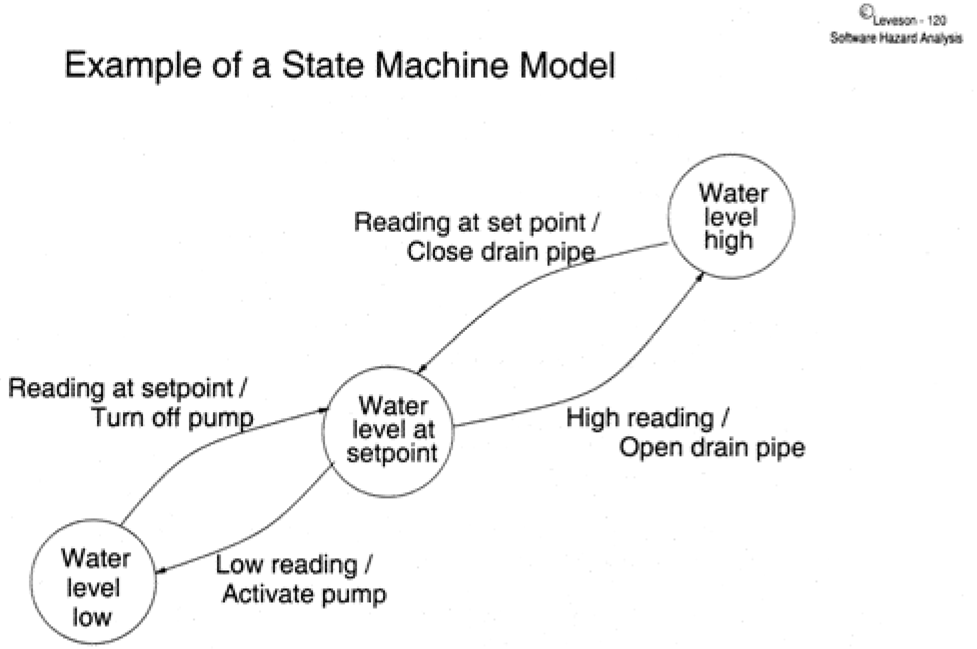
System and Software Models

- Representation of the parts and behaviour of a solution
- Used during ideation to clarify ideas
- Used to communicate specifics to stakeholders
- Used to evaluate solution quality
- Used to unify systems during construction
- Used as a maintenance reference
- Used as a guide for future work
    - Similar situations can use models as a pattern
    - Future generations of the software can use models as a starting point

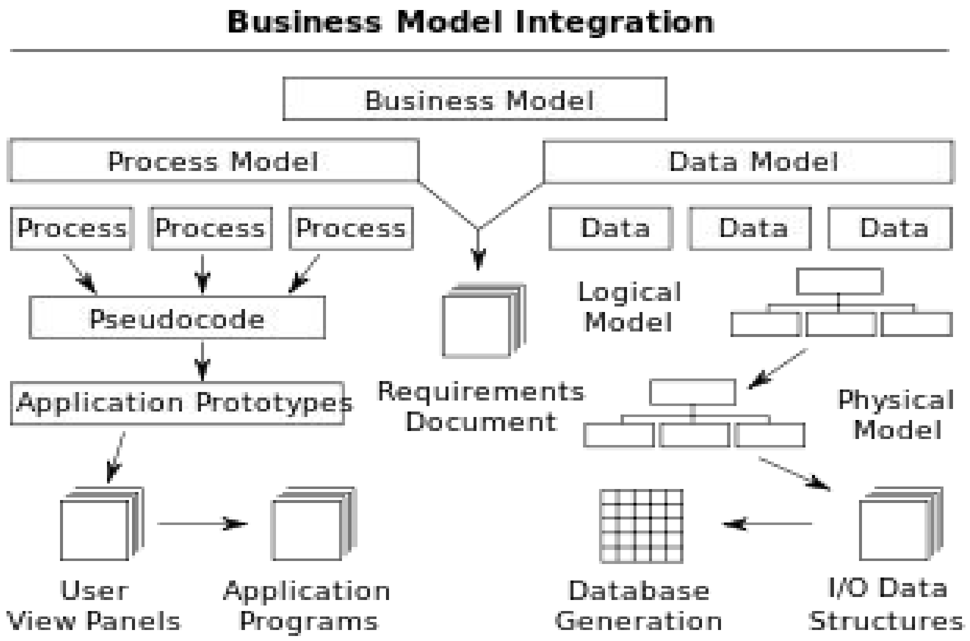
A Model of Software



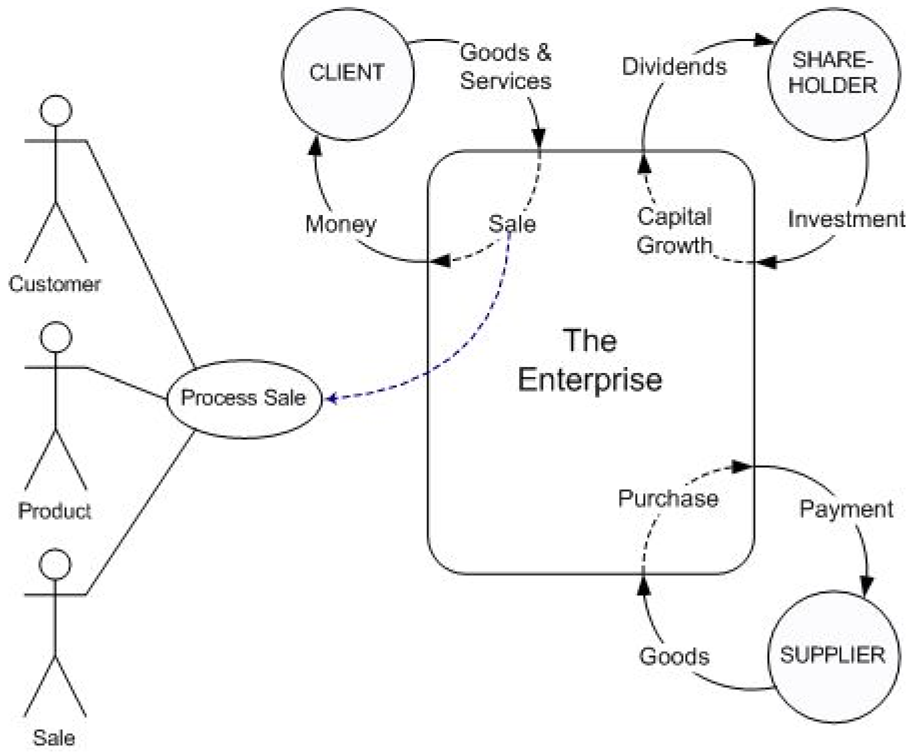
State Models


©Leveson - 120
Software Hazard Analysis
Example of a State Machine Model
Water level high
Reading at set point / Close drain pipe
Reading at setpoint / Turn off pump
Water level at setpoint
High reading / Open drain pipe
Water level low
Low reading / Activate pump

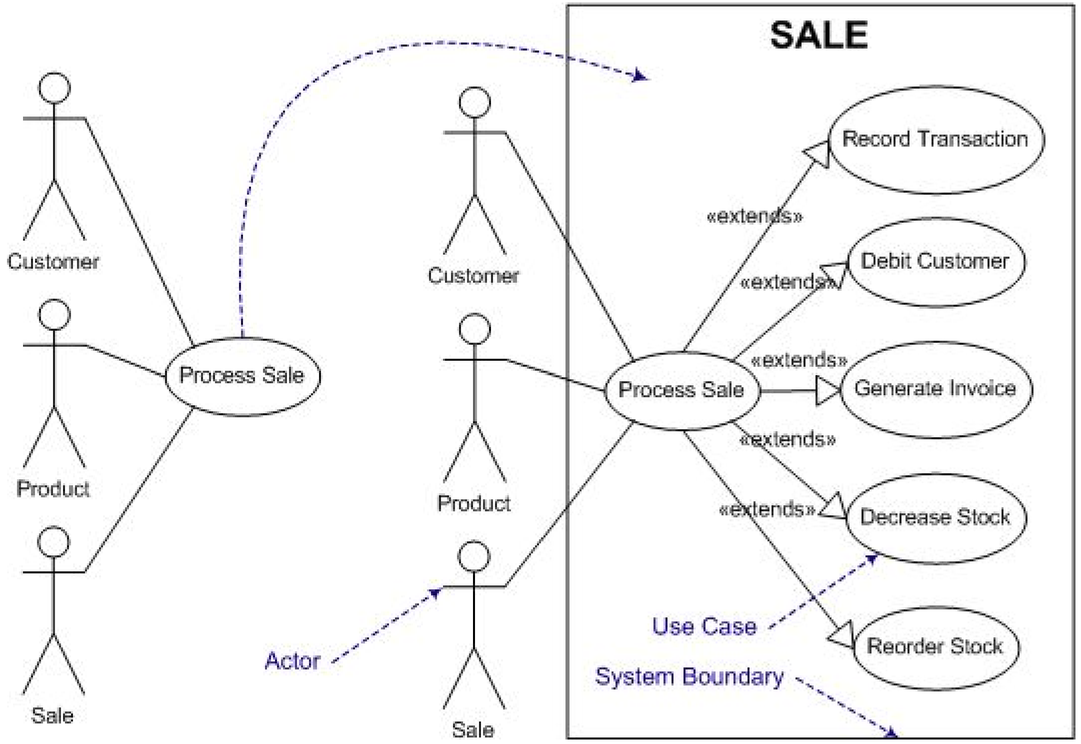
Business Models



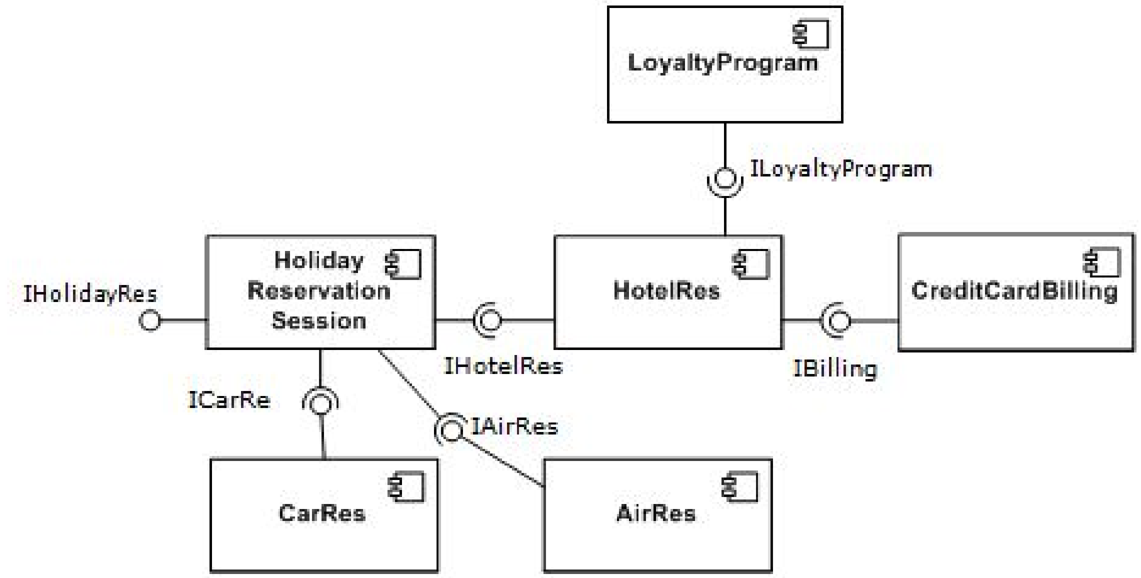
Business Process Model



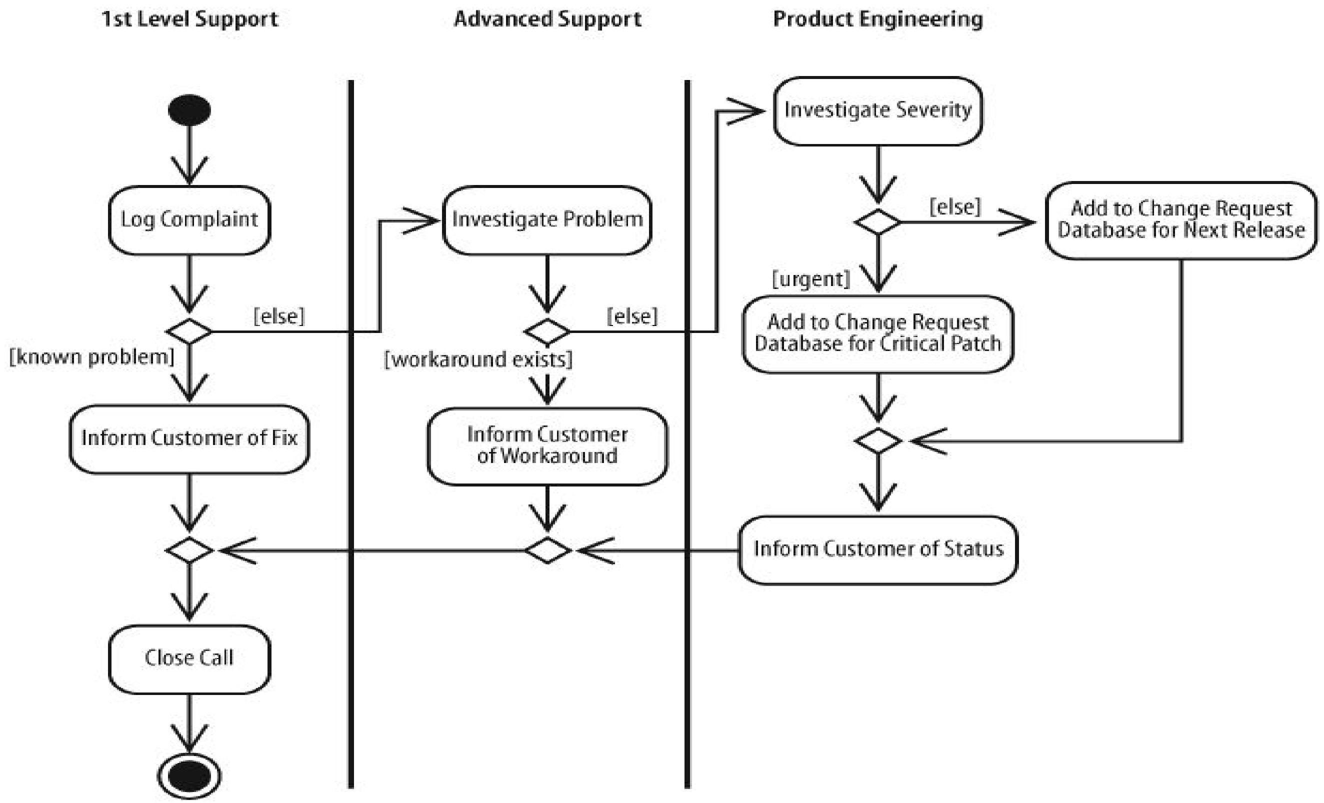
Use Case Model



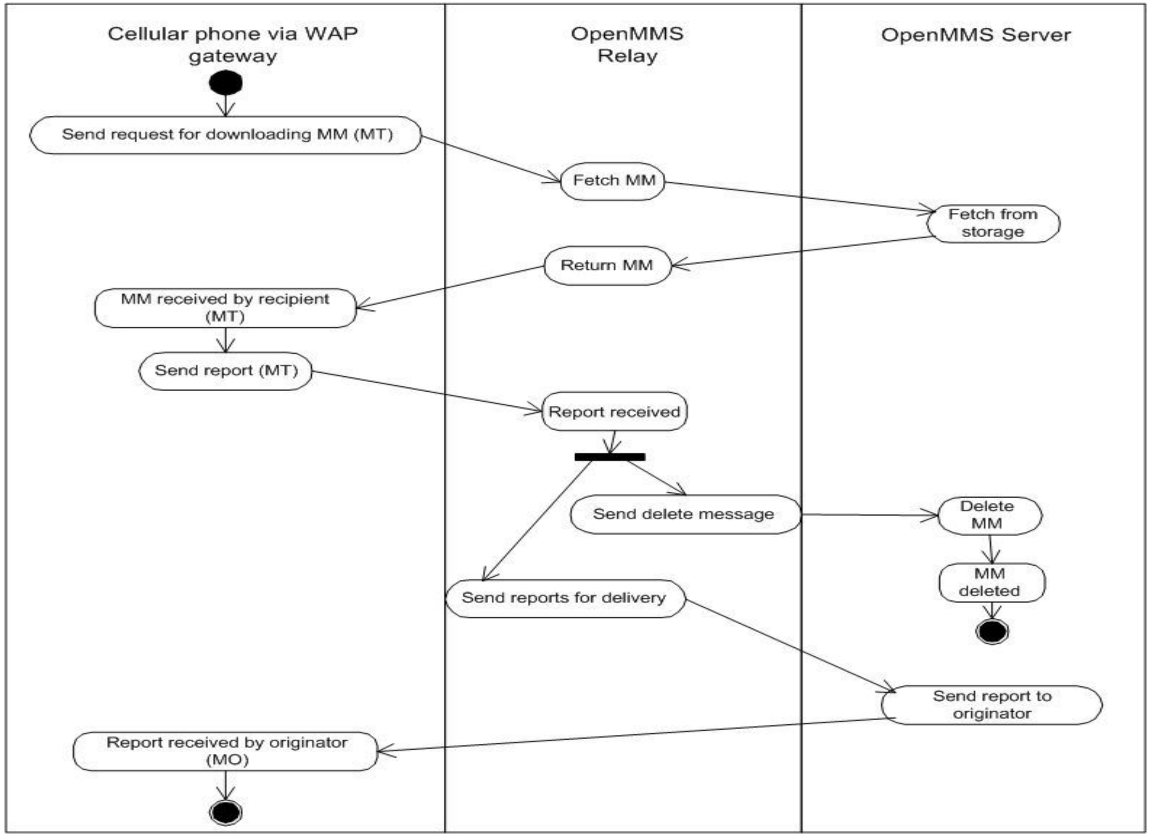
Software Component Model



System Interaction Model (Swim Lane)

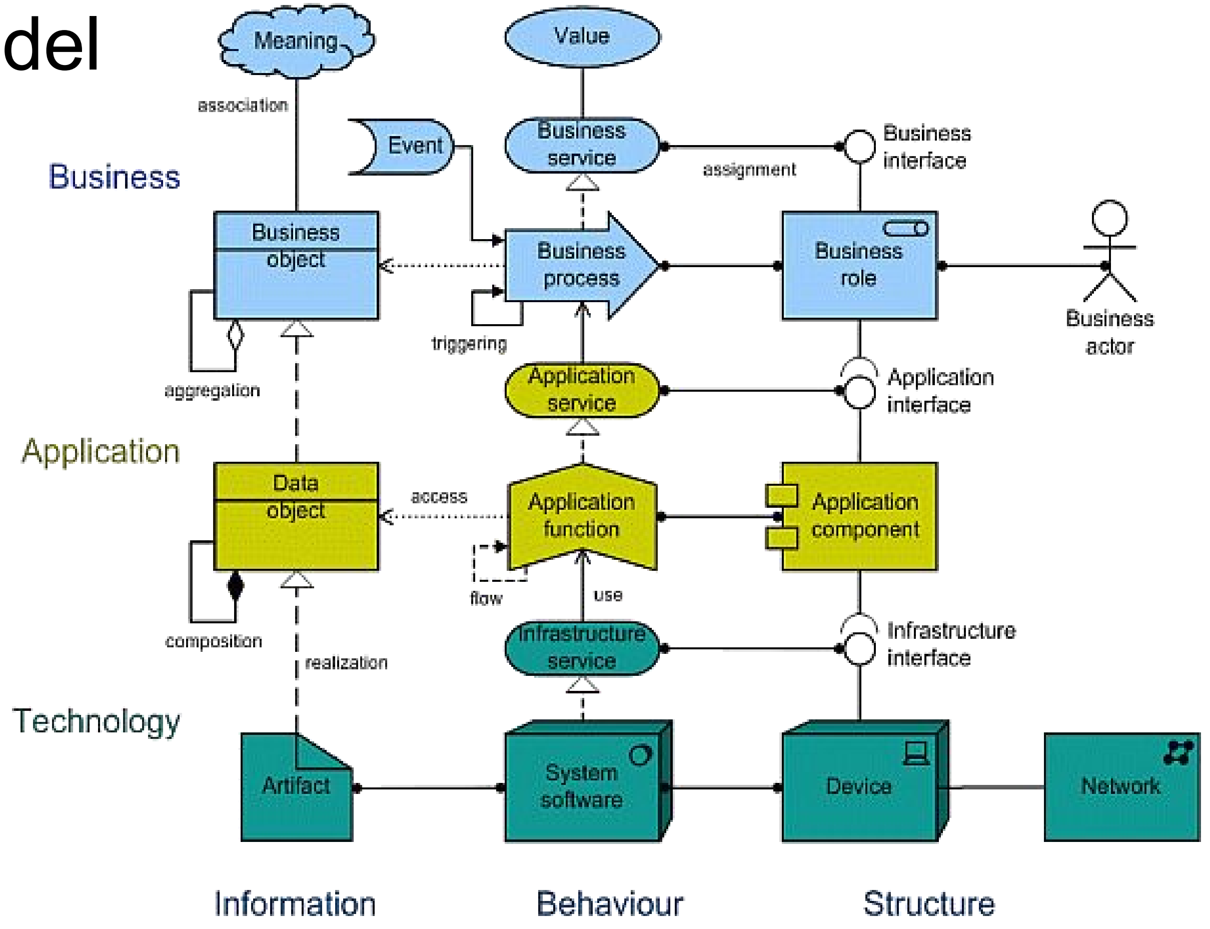


System Interaction Model (Swim Lane)

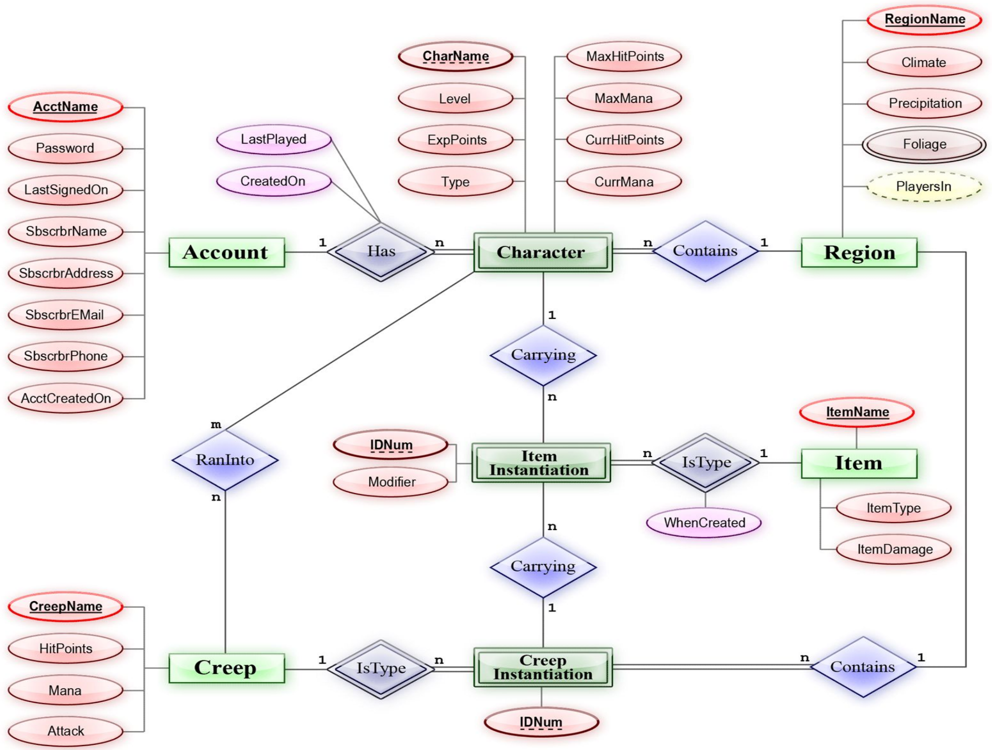

Cellular phone via WAP gateway
OpenMMS Relay
OpenMMS Server
Send request for downloading MM (MT)
Fetch MM
Fetch from storage
Return MM
MM received by recipient (MT)
Send report (MT)
Report received
Send delete message
Delete MM
MM deleted
Send reports for delivery
Send report to originator
Report received by originator (MO)

Architectural Model

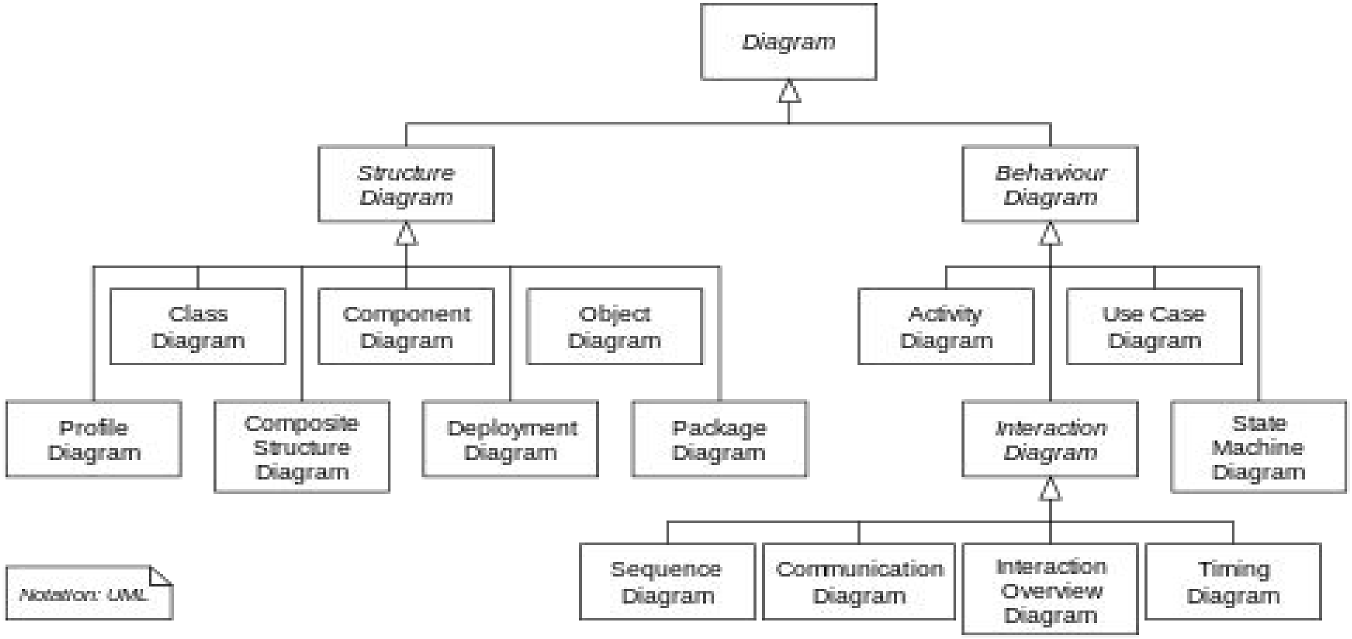
This Archimate drawing is drawn by the free "Archi" tool -- more about this later.



Entity-Relationship Model



Structure - Diagram Hierarchy



Graphical Modeling Languages

- UML
  - *https://en.wikipedia.org/wiki/Unified_Modeling_Language*
- Archimate
  - *http://www.opengroup.org/subjectareas/enterprise/archimate*
  - *http://www.archimatetool.com/*
- ERD
  - *https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model*
  - *http://erwin.com/*

Demo Time

Web Sites of Interest

- http://creately.com/Draw-UML-and-Class-Diagrams-Online
- http://www.archimatetool.com/
- https://www.lucidchart.com/pages/landing/uml_diagram_tool
- https://www.websequencediagrams.com/
- http://www.opengroup.org/subjectareas/enterprise/archimate