SE463
Software Requirements Specification & Analysis

Prioritizing Requirements

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Prioritizing Requirements

- There are often more requirements than can be implemented
  - Customers ask for way too much
  - Need to balance requirements against limitations in budget, staff, schedule
  - Need to decide which features go into the next release

- Requirements triage
  - Some requirements must be implemented
  - Some requirements may be obvious choices to exclude
  - The rest are desirables that we need to select from

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Prioritization Aspects

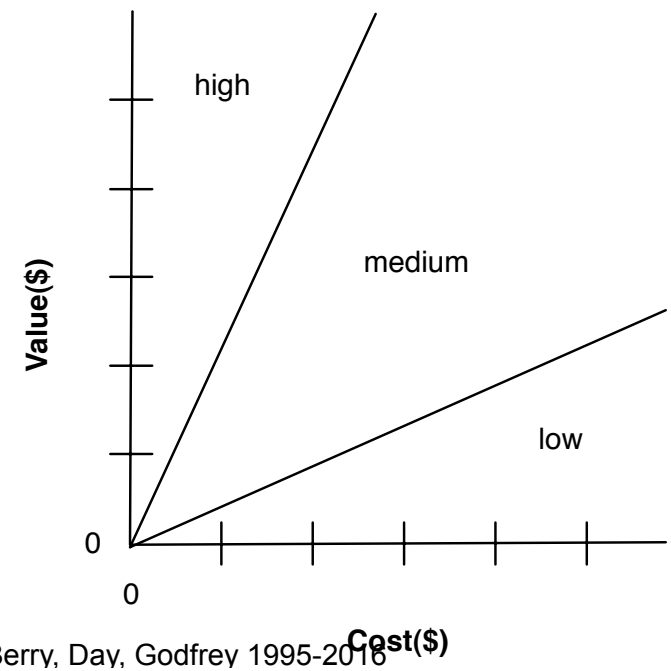
- Business Value Added
- Penalty / Harm Avoided
- Risk
- Cost
- Time

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Combining Aspects

Most companies prioritize requirements by their potential value and cost.

- Value is a requirement's potential contribution to customer satisfaction
- Cost is the cost of implementing the requirement
- Can prioritize requirements according to their cost-value ratios



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Grouping Requirements

The most common prioritization technique is numerical assignment or grouping of requirements into 3-4 priority groups.

- Critical
- Standard
- Optional

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

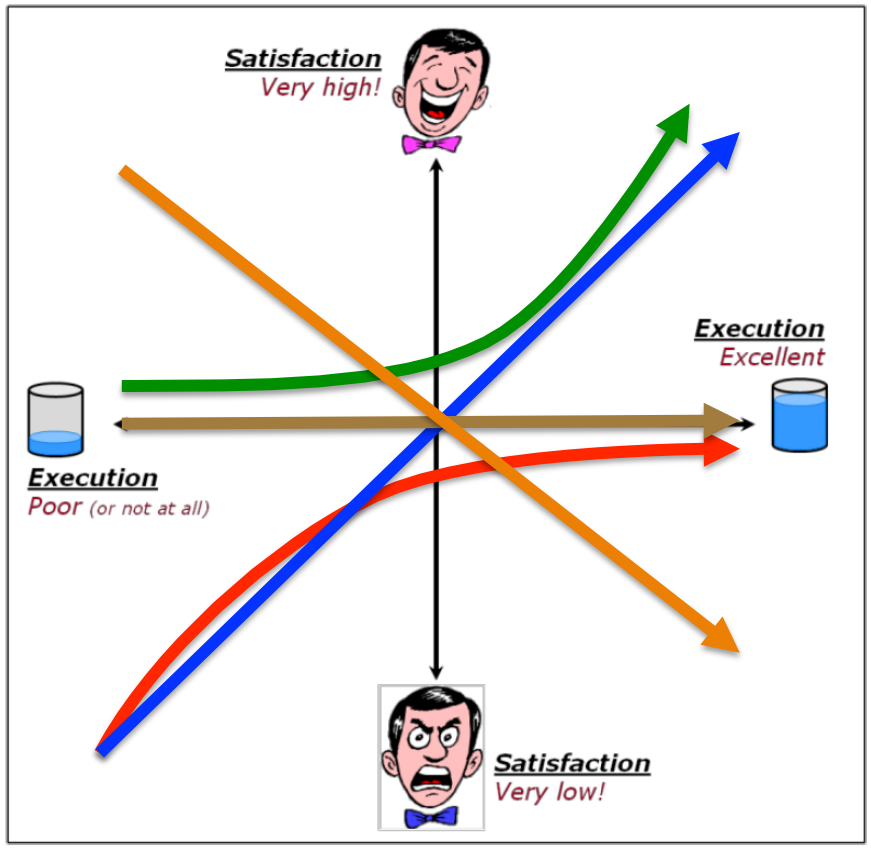
Ranking Requirements

Each requirement is assigned a unique rank (1, 2, …), but it is not possible to see the relative difference between ranked requirements.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Kano Model

The Kano Model is a method for grouping requirements based on customer perception, in order to select the requirements that deliver the greatest customer satisfaction.



Basic: requirements that the customer takes for granted

Performance: requirements that the customer specifically asked for

Excitement: requirements that the customer does not request or expect

Indifferent: requirements that the customer does not care about

Reverse: “requirements” that the customer hates

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Kano Surveys

1. Ask customers what their reaction would be if the requirement were included in the product
2. Ask customers what their reaction would be if the requirement were NOT included in the product.

The possible answers are

- I like it
- I expect it
- I am neutral
- I can tolerate it
- I dislike it

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Kano Surveys

Customer Survey Responses		Disfunctional Question Answer				
		Like	Expect	Neutral	Tolerate	Dislike
Functional Question Answer	Like	*Questionable*	Excitement	Excitement	Excitement	Performance
	Expect	Reverse	*Questionable*	Indifferent	Indifferent	Basic
	Neutral	Reverse	Indifferent	Indifferent	Indifferent	Basic
	Tolerate	Reverse	Indifferent	Indifferent	*Questionable*	Basic
	Dislike	Reverse	Reverse	Reverse	Reverse	*Questionable*

Basic > Performance > Excitement > Indifferent

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

100-Dollar Test

In cumulative voting, or the 100-dollar test, stakeholders are given 100 prioritization points (votes) to distribute among the requirements.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Analytic Hierarchy Process

Requirements prioritization technique that analyzes stakeholders' pairwise comparisons of requirements, and produces a relative ranking of requirements.

- Acknowledges that absolute values and costs are hard to estimate
- Relative comparisons are easier

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

AHP Pairwise Comparisons

Compare pairs of requirements

- 1 - requirements are of equal value
- 3 - one is slightly preferred over the other
- 5 - one is strongly preferred over the other
- 7 - one is very strongly preferred over the other
- 9 - one is extremely preferred over the other
- intermediate values 2, 4, 6, 8 used when compromise is needed
- if pair (x,y) has relative value n, complementary pair (y,x) has reciprocal value 1/n

	Req1	Req2	Req3	Req4
Req1	1	1/3	2	4
Req2	3	1	5	3
Req3	1/2	1/5	1	1/3
Req4	1/4	1/3	3	1

Copyright © Atlee, Berry, Day, Godfrey 199

AHP Analysis

Estimate eigenvalues by "averaging over normalized columns"

Step 1: Compare pairs of requirements

	Req1	Req2	Req3	Req4
Req1	1	1/3	2	4
Req2	3	1	5	3
Req3	1/2	1/5	1	1/3
Req4	1/4	1/3	3	1

Step 2: Normalize the columns
(i.e., divide each entry by the sum of its column

	Req1	Req2	Req3	Req4
Req1	0.21	0.18	0.18	0.48
Req2	0.63	0.54	0.45	0.36
Req3	0.11	0.11	0.09	0.04
Req4	0.05	0.18	0.27	0.12

Step 3:
Sum each row

Sum
1.05
1.98
0.34
0.62

Step 4:
Normalize sums

Sum/4
0.26
0.50
0.09
0.16

Step 5:
Report relative values

Req1	26%
Req2	50%
Req3	9%
Req4	16%

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

AHP Checking Consistency

Step 1: Multiply comparison matrix by priority vector

	Req1	Req2	Req3	Req4
Req1	1	1/3	2	4
Req2	3	1	5	3
Req3	1/2	1/5	1	1/3
Req4	1/4	1/3	3	1

•

Priority
0.26
0.50
0.09
0.16

=

1.22
2.18
0.37
0.64

Step 2: Divide each element by the corresponding element in priority vector

1.22 / 0.26
2.18 / 0.50
0.37 / 0.09
0.64 / 0.16

=

4.66
4.40
4.29
4.13

Step 3: Compute principle eigenvalue

$$\frac{4.66 + 4.40 + 4.29 + 4.13}{4} = 4.37$$

Step 4: Calculate consistency index

$$CI = \frac{4.37 - n}{n-1} = 0.12$$

Step 5: Compare against consistency index of random matrix (<0.10)

$$CR = \frac{0.12}{0.90} = 0.14$$

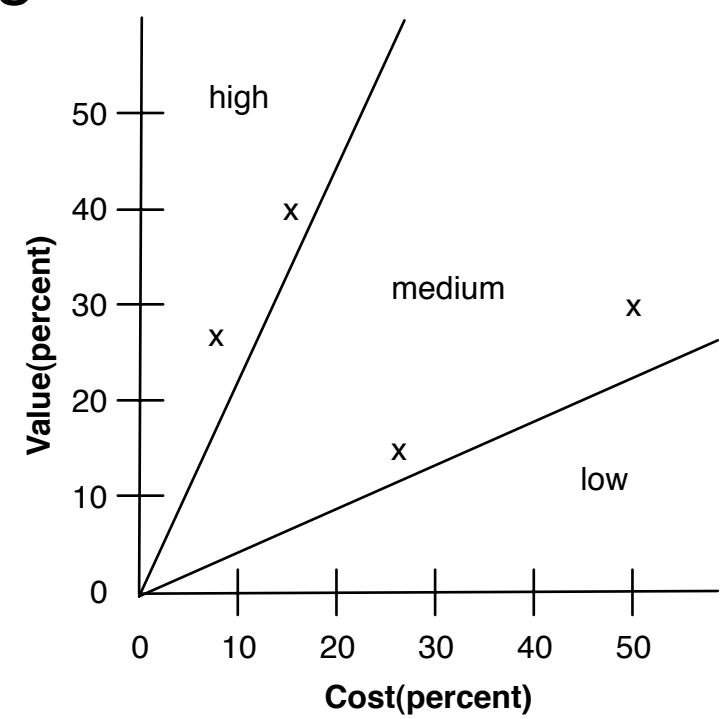
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Combining Different Techniques

Categorize requirements into groups, and then prioritize within each group

Plot return-on-investment ratios

- Relative value / relative cost



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Challenges

- All requirements deemed to be essential
- Large number of requirements to prioritize
- Conflicting priorities
- Changing priorities
- Stakeholder and developer collaboration
- Subjective prioritization

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Benefits

- Improves customer satisfaction, by implementing most important requirements first
- Helps to determine how to prioritize the allocation of limited project resources
- Encourages stakeholders to consider all requirements (not just their own)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

- Purposes of prioritization
- Aspects of prioritization
- Prioritization techniques
- How techniques vary with respect to their efficiency, precision, consistency
- Benefits and challenges of prioritizing requirements

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Deliverable #6

- 100-Dollar Prioritization of all Use Cases
  - with respect to importance/value
  - by 4 stakeholders, 3 not on your team
- Kano Prioritization of Scenarios (main, alternatives, and exceptions) of the two highest-priority Use Cases
  - with respect to importance/value
  - by 4 stakeholders, 3 not on your team
- 100-Dollar Prioritization of Atomic Requirements of the highest-priority Scenarios
  - with respect to relative importance/value
  - by 2 stakeholders not on your team
- AHP Prioritization of Atomic Requirements of the highest-priority Scenarios
  - with respect to relative difficulty to implement
  - by all members your team
- Plot relative value / relative difficulty of Atomic Requirements

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Business Rules, OCL

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Business Rules

A business rule is an assertion that defines or constrains some aspect of the Work.

Examples:

- Rental agreements must be no longer than 4 weeks long
- A customer must be at least 25 years old
- A customer cannot rent more than 3 red cars

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

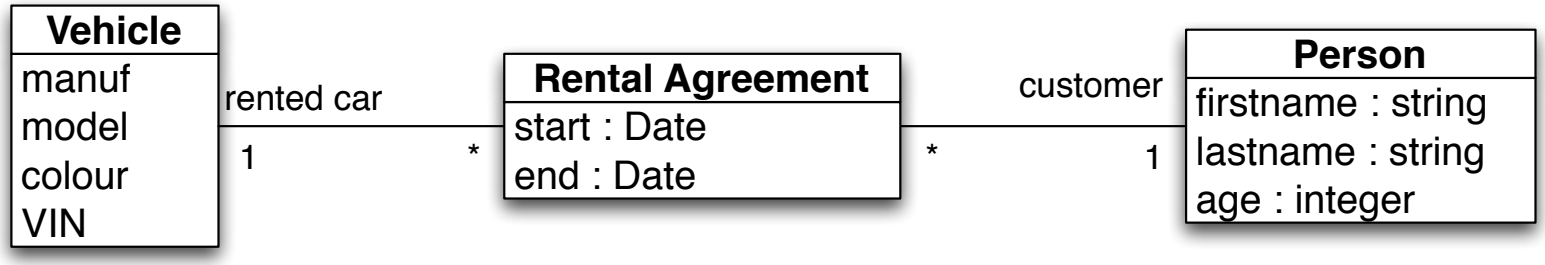
Object Constraint Language

- Complements the UML
  - standardized by the Object Management Group (OMG)
  - not one of the UML notations
  - used to express constraints on UML models
  - precise, yet (relatively) easy to read

- It has language constructs for
  - relating classes that have no direct association
  - expressing queries over objects and collections of objects

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

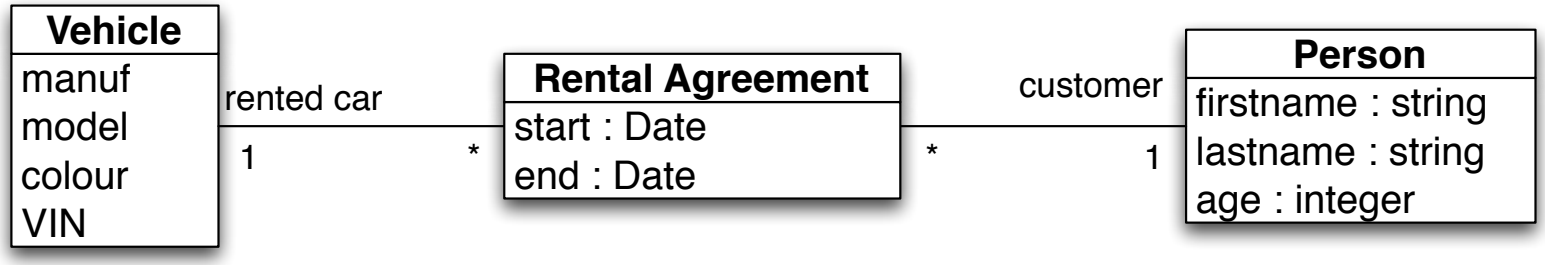
Simple OCL example



Rental agreements must be less than 4 weeks long

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

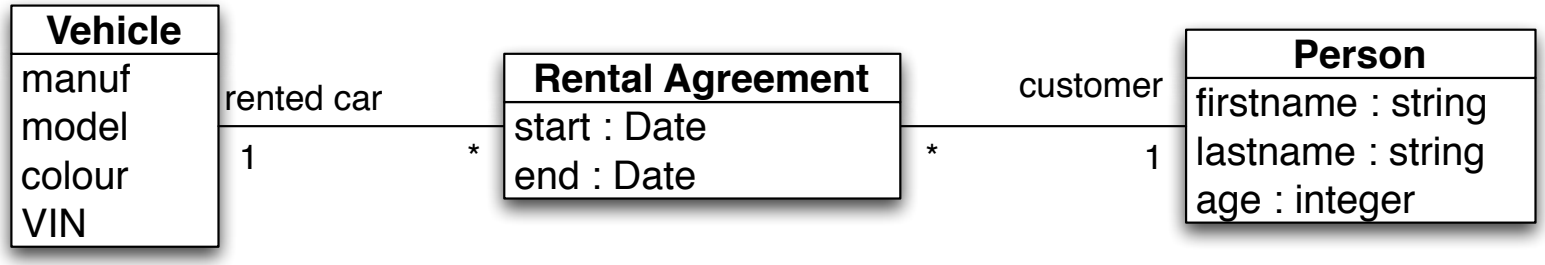
Another example



A customer must be at least 25 years old

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Constraint Expressions

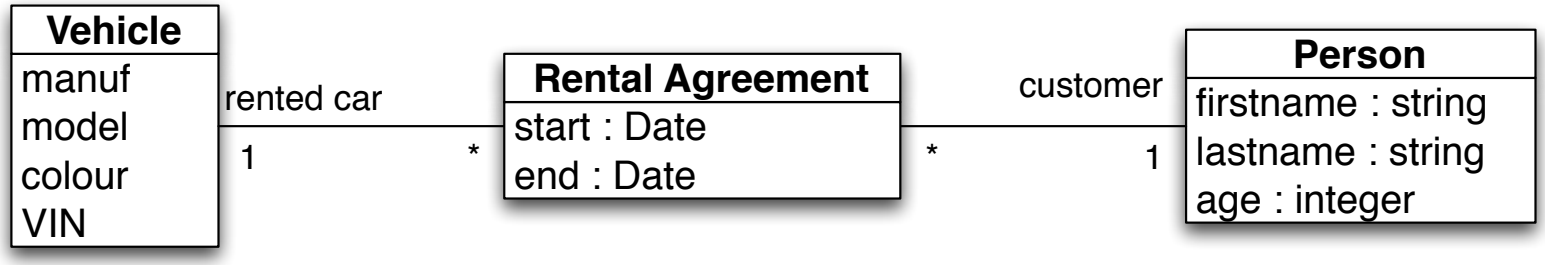


Expressions over

- attributes
- navigations derived from associations (and aggregations)
- rolename on far end of association
- class name on far end of association
- literal values

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

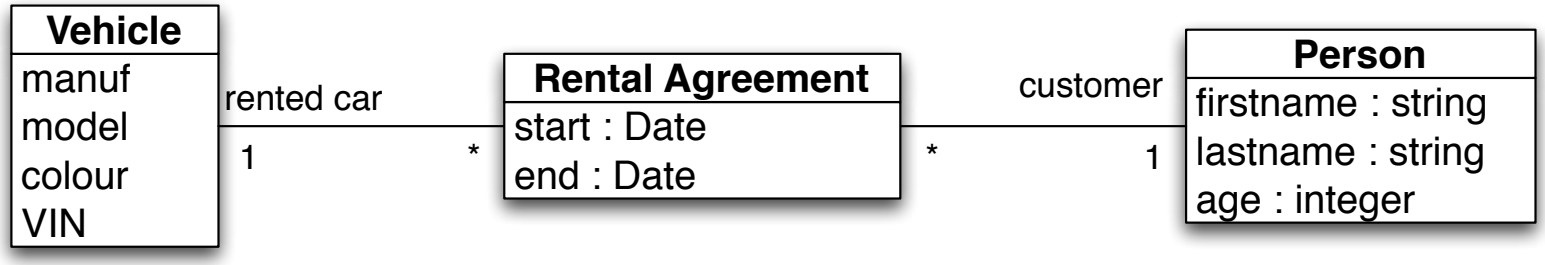
A third example



A customer cannot rent more than 3 cars

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Navigation Across Associations



Consider object p:Person:

Expression	Value
p	
p.RentalAgreement	
p.RentalAgreement.rented_car	
p.RentalAgreement.rented_car.colour	

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Navigation Across Associations

Example model	Navigation expressions: Expression	Navigation expressions: Value
A (a1:String) —b— 1 — 1 —c— B (b1:String) —c— 1 — 1 — C (c1:String); context	self	The contextual instance – an instance of A
	self.b	An object of type B
	self.b.b1	The value of attribute B::b1
	self.b.c	An object of type C
	self.b.c.c1	The value of attribute C::c1
D (d1:String) —e— 1 — 1 —e— E (e1:String) —f— * — * —f— F (f1:String); context	self	The contextual instance – an instance of D
	self.e	An object of type E
	self.e.e1	The value of attribute E::e1
	self.e.f	A Set(F) of objects of type F
	self.e.f.f1	A Bag(String) of values of attribute F::f1
G (g1:String) —h— * — * —h— H (h1:String) —i— 1 — 1 —i— I (i1:String); context	self	The contextual instance – an instance of G
	self.h	A Set(H) of objects of type H
	self.h.h1	A Bag(String) of values of attribute H::h1
	self.h.i	A Bag(I) of objects of type I
	self.h.i.i1	A Bag(String) of values of attribute I::i1
J (j1:String) —k— * — * —k— K (k1:String) —l— * — * —l— L (l1:String); context	self	The contextual instance – an instance of J
	self.k	A Set(K) of objects of type K
	self.k.k1	A Bag(String) of values of attribute K::k1
	self.k.l	A Bag(L) of objects of type L
	self.k.l.l1	A Bag(String) of values of attribute L::l1

Figure 25.12

Arlow and Neustadt, *UML 2 and the Unified Process*

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Operations on Collections

OCL operations on collections are prefaces with an arrow notation (->), to distinguish from model-defined names or operations

// size() refers to (nonexistent) method ”size" in class Vehicle

context Person **inv:**

self.RentalAgreement.Vehicle.size() <= 3

// size() is OCL operation that counts number of vehicles

context Person **inv:**

self.RentalAgreement.Vehicle->size() <= 3

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Basic types

Booleans		
Operation	Notation	Result Type
or	a or b	Boolean
and	a and b	Boolean
exclusive or	a xor b	Boolean
negation	not a	Boolean
equals	a = b	Boolean
not equals	a <> b	Boolean
implies	a implies b	Boolean
if then else	if a then b else b'	Type of b and b'

Strings		
Operation	Notation	Result Type
concatenation	s.concat(t)	String
size	s.size	Integer
to lower case	s.toLower	String
to upper case	s.toUpper	String
substring	s.substring(int,int)	String
equals	s=t	Boolean
not equals	s<>t	Boolean

Integers and Reals		
Operation	Notation	Result Type
equals	a = b	Boolean
not equals	a <> b	Boolean
less	a < b	Boolean
more	a > b	Boolean
less or equal	a <= b	Boolean
more or equal	a >= b	Boolean
plus	a + b	Integer or Real
minus	a - b	Integer or Real
multiplication	a * b	Integer or Real
division	a / b	Real
modulus	a.mod(b)	Integer
integer division	a.div(b)	Integer
absolute value	a.abs	Integer or Real
maximum value	a.max(b)	Integer or Real
minimum value	a.min(b)	Integer or Real
round	a.round	Integer
floor	a.floor	Integer

Arlow and Neustadt, *UML 2 and the Unified Process*

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Expressions on collections

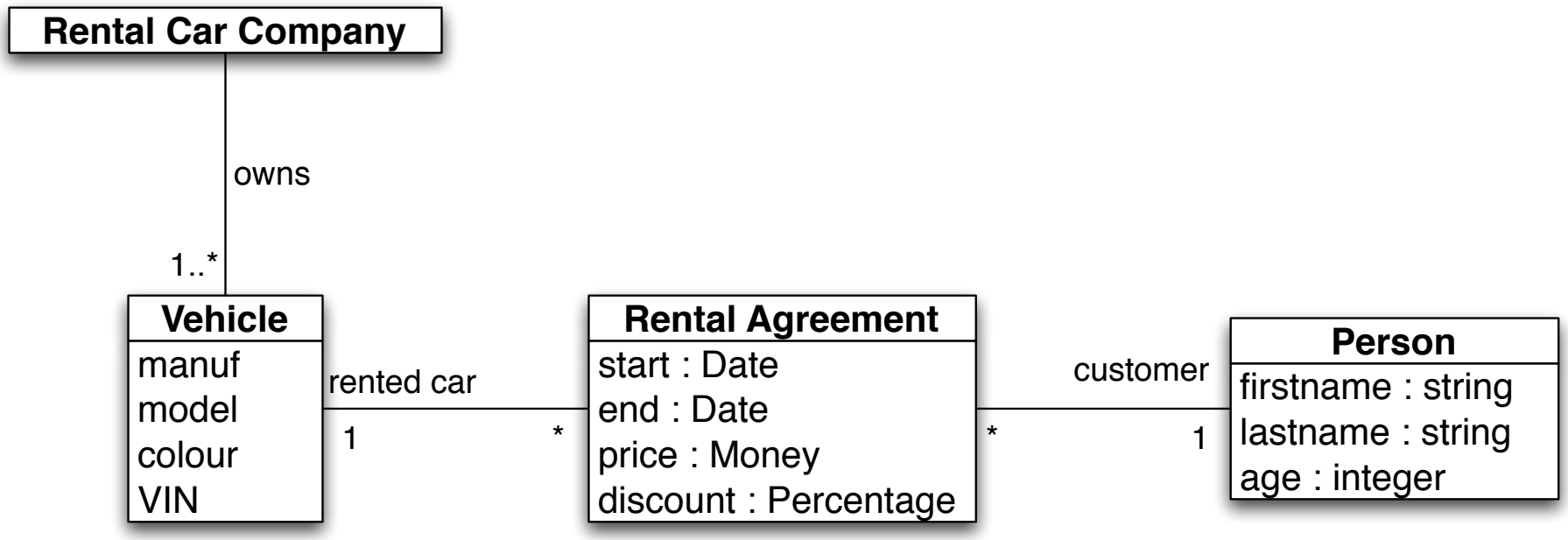
Set of T		
Operation	Notation	Result Type
equals	a = b	Boolean
not equals	a <> b	Boolean
size	a–>size()	Integer
sum	a–>sum()	Type T
count	a–>count(t)	Integer
includes	a–>includes(t)	Boolean
excludes	a–>excludes(t)	Boolean
includes all	a–>includesall(b)	Boolean
excludes all	a–>excludesall(b)	Boolean
is empty	a–>isEmpty()	Boolean
not empty	a–>notEmpty()	Boolean
union	a–>union(b)	Set of T
intersection	a–>intersection(b)	Set of T
difference	a - b	Set of T
insert	a–>including(t)	Set of T
remove	a–>excluding(t)	Set of T

a, b : Set of Type T
t : object of Type T

Arlow and Neustadt, *UML 2 and the Unified Process*

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

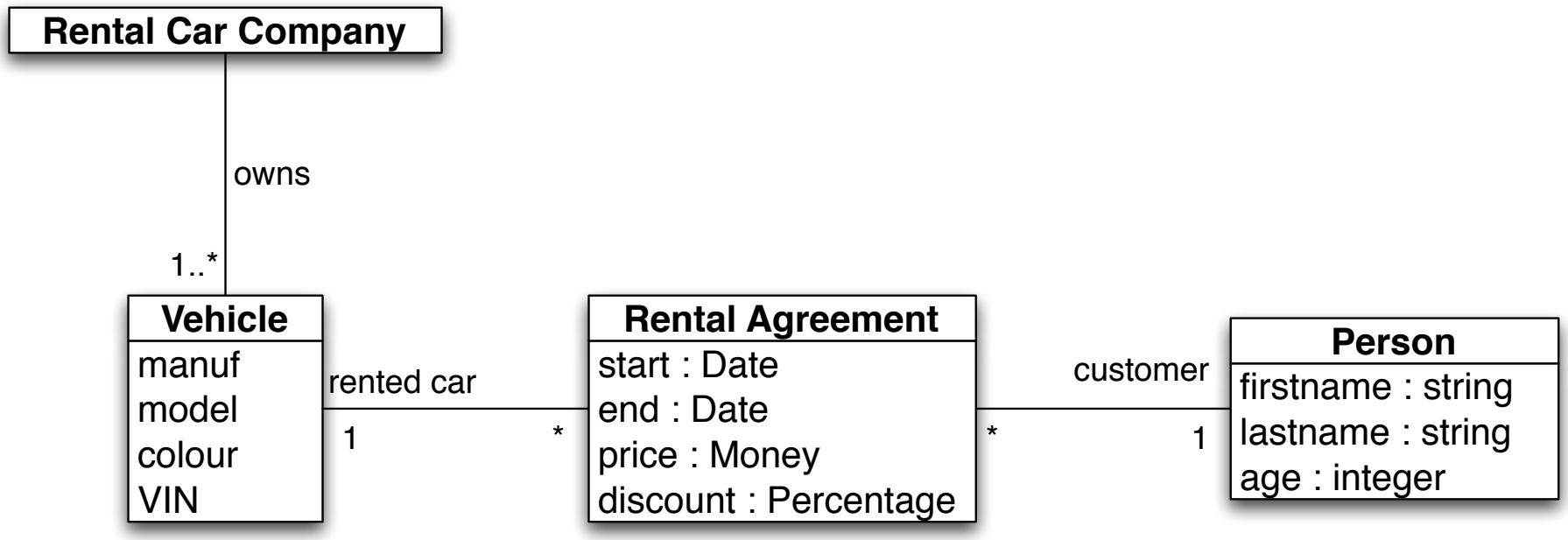
More Examples



Rental car companies never own red cars

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

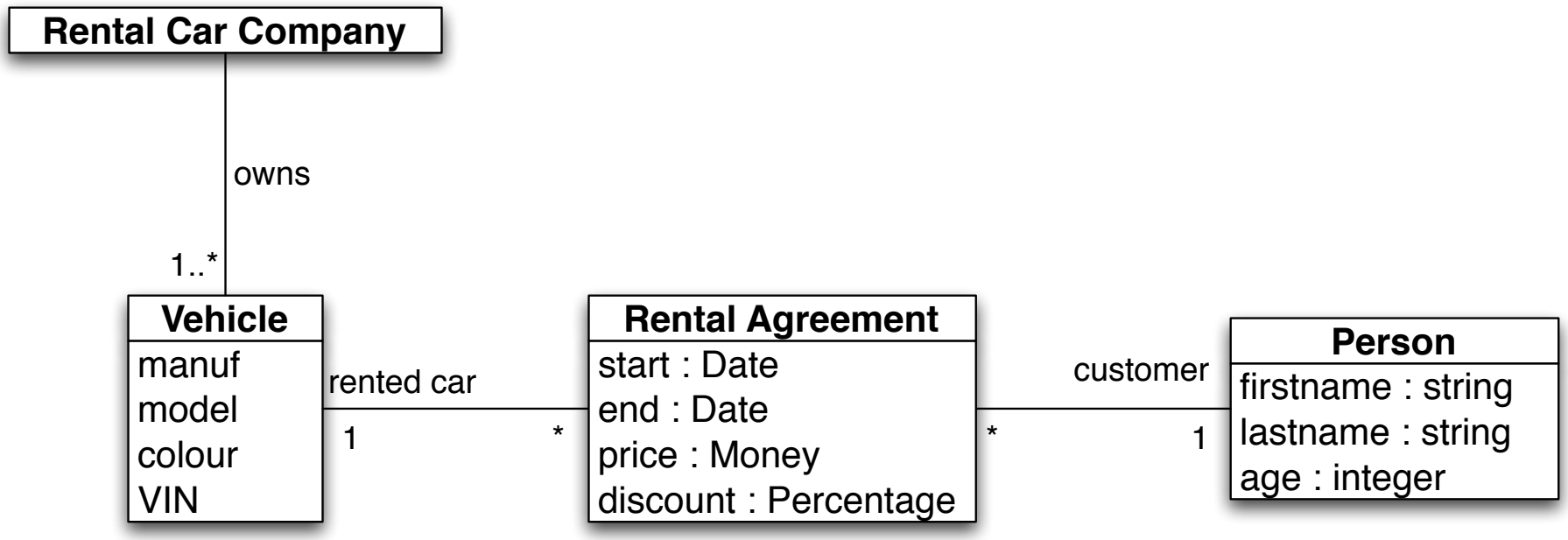
More Examples



Frequent renters (# rentals > 20) receive 10% discount

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

More Examples



Heavy spenders (sum of rentals > $5000) receive 20% discount

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Filtering Operations

There are operations extracting specific elements from an existing collection, based on the value of an expression.

- select returns the elements that satisfy given expression
- reject returns the elements that falsify given expression

Rental car companies never own red cars

context RentalCarCompany **inv:**
self.owns->select(colour="red")->isEmpty()

Customers younger than 30 cannot rent red cars

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Quantification

The exists operation is used to assert that at least one element in a collection satisfies some expression. The operation returns a boolean value.

Every customer rents at least one black car

context Person **inv:**

self.RentalAgreement.Vehicle->exists(colour="black")

Every car is being rented today

(i.e., every car is involved in some rental agreement today)

context Vehicle **inv:**

self.RentalAgreement->exists(r : RentalAgreement |

    r.start <= today and today <= r.end)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Quantification

The forAll operation is used to assert that all members of a set satisfy a given expression. The operation returns a boolean.

All rental cars are white

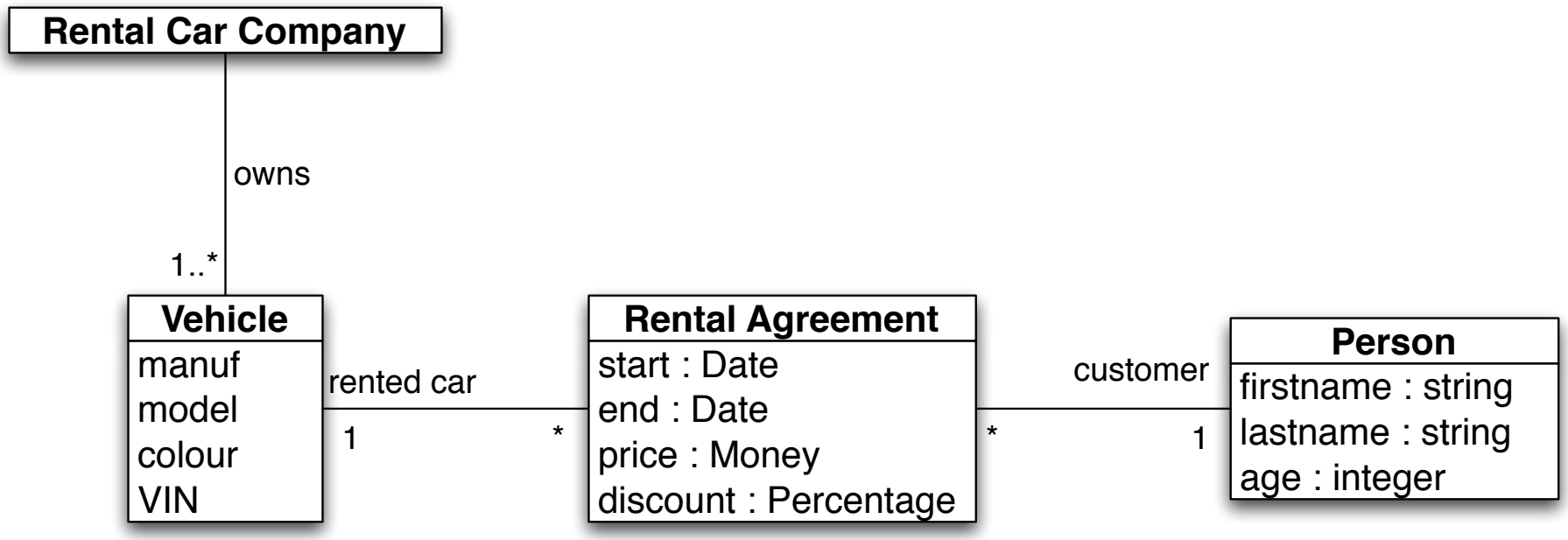
context RentalCarCompany **inv:**
self.owns->forall(colour="white")

No car is rented more than once each day

context Vehicle **inv:**
self.RentalAgreement->forall(r1, r2 : RentalAgreement |
    (r1 <> r2) implies (r1.start > r2.end or r2.start > r1.end))

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

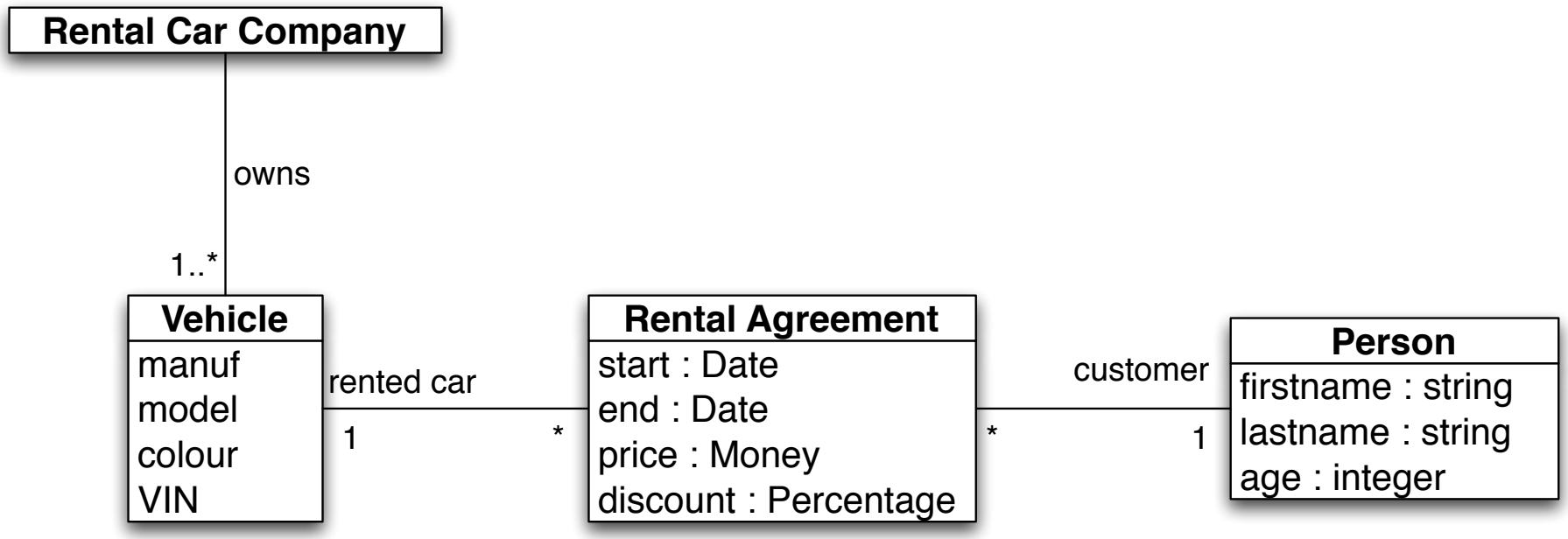
More Examples



No rental agreement has a price of less than $20

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

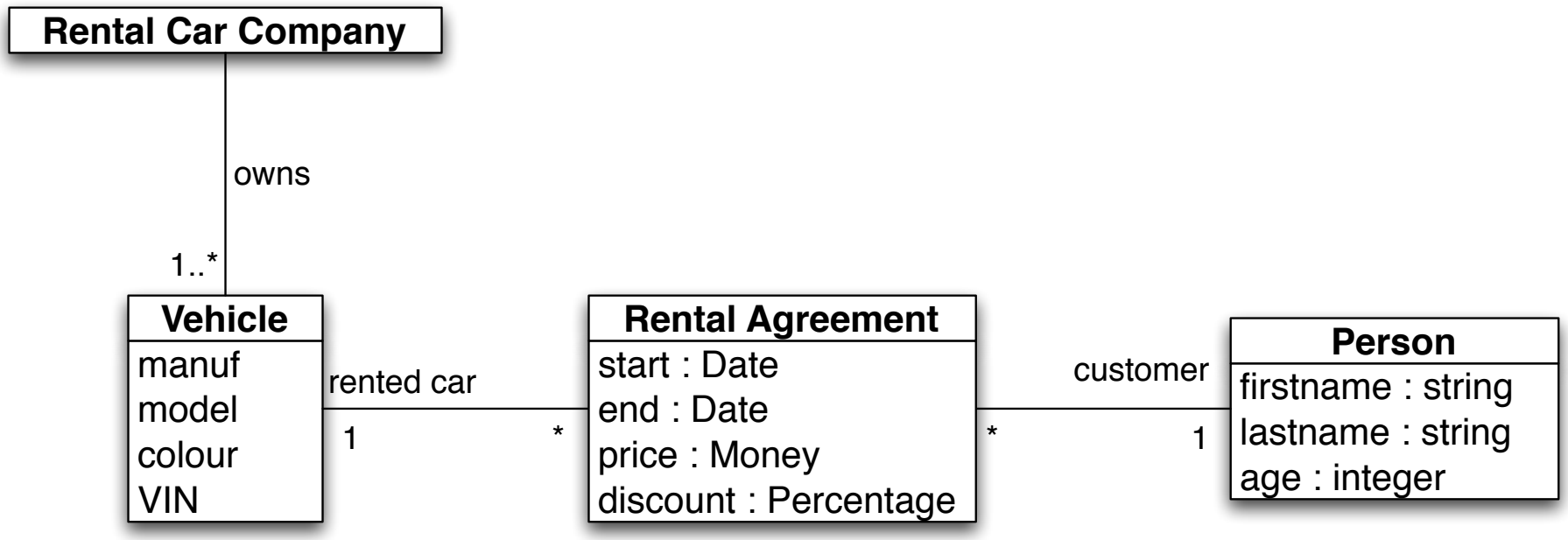
More Examples



The rental car fleet contains at least one Toyota

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

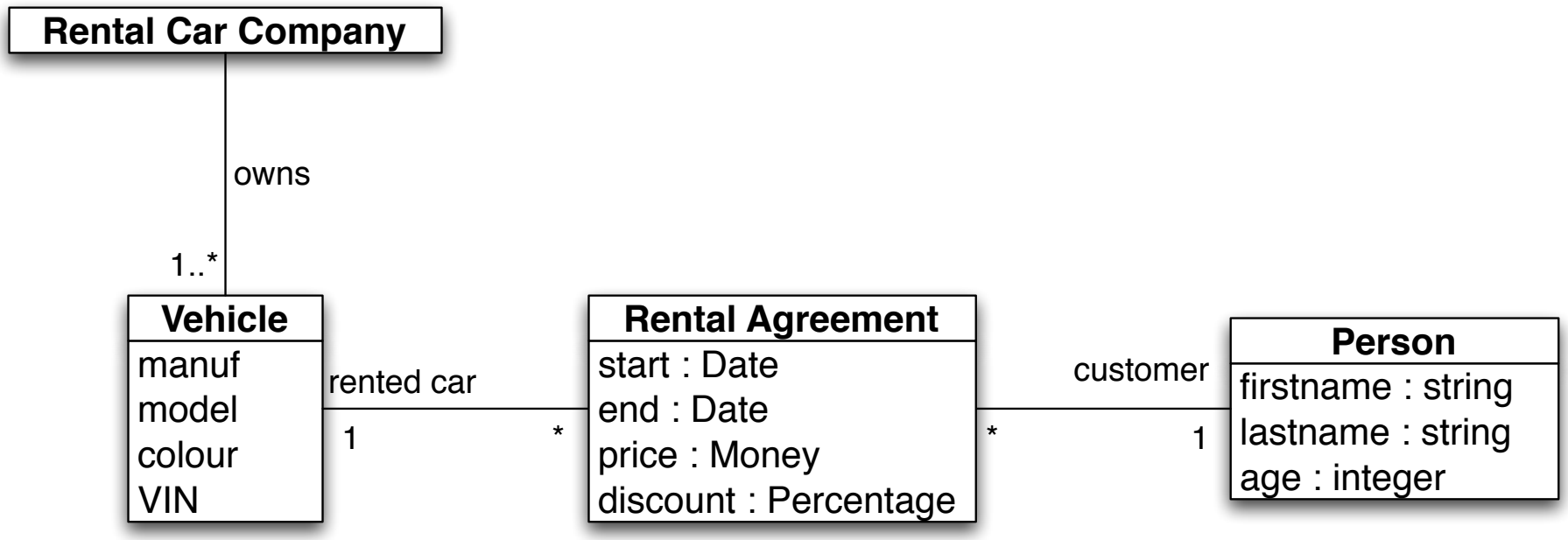
More Examples



Red cars are never rented for less $100/day

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

More Examples



No person rents the same car twice

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Broken Rules

Note that business rules simply state what ought to be true. If the execution of the system leads to a case where the rule is not true, we say that the rule is broken or violated.

There is nothing in the description of a business rule that says how to recover from a broken rule.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

OCL Tools

There are a number of tools that support OCL, from both universities and industry. These tools range from

- Parsers and type checkers
- Evaluators that can check an OCL expression against all instances of a UML class model
- Debuggers that step through an OCL expression and check each subsection (to locate faulty subexpression)
- Code generators that translate OCL expressions into run-time assertions

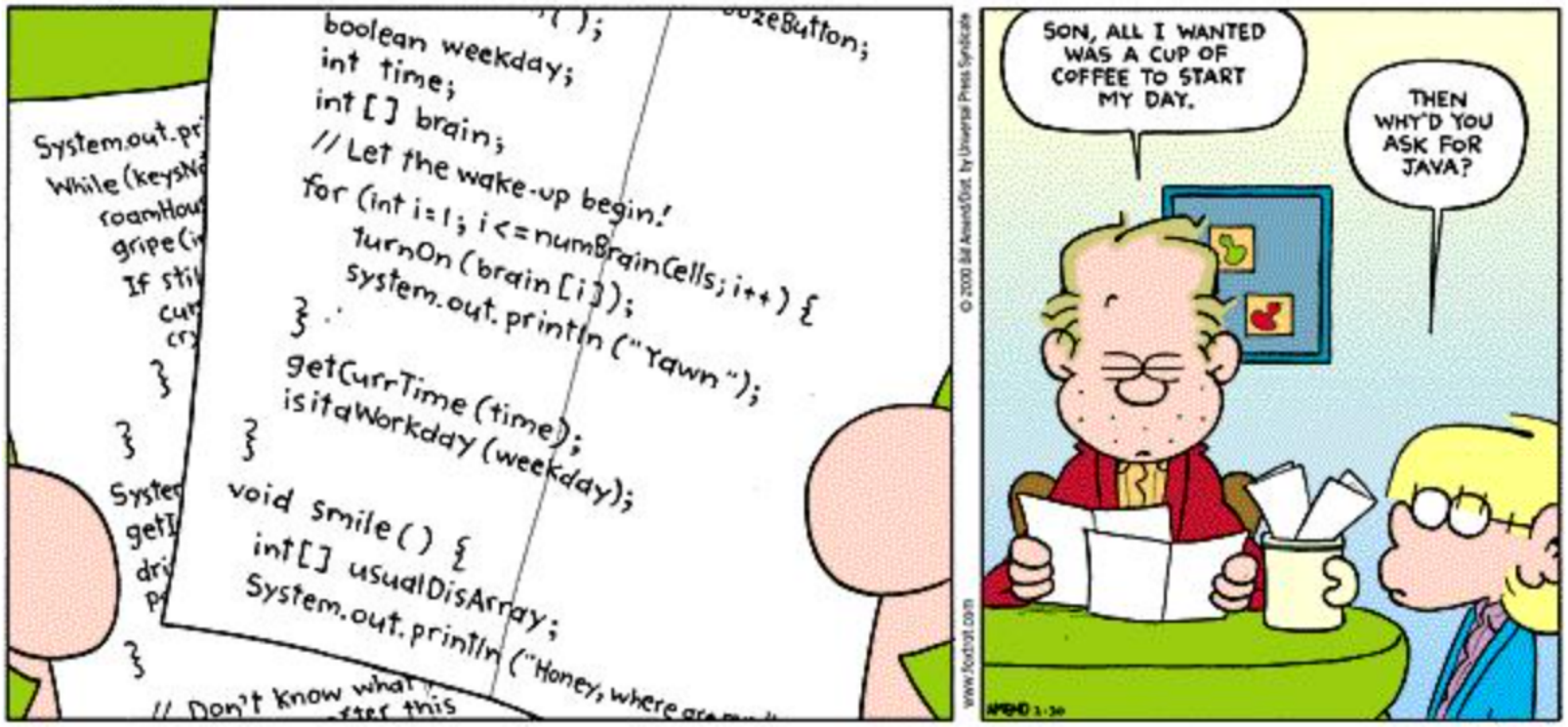
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Object Constraint Language (OCL)

- augments UML
- expresses business rules as constraints on the domain model

Copyright © Atlee, Berry, Day, Godfrey 1995-2016



© 20, February, 2000, Bill Amend, Distributed by University Press Syndicate

Updated Use Case Diagram

- Mostly done well.
- Some Use Case Diagrams still model sub use cases rather than comprehensive business use cases
- Important to get this right for Deliverable #6
- Ask your TA about this if you did not get 10/10 on level of detail

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Activity Diagram

- Mostly done well
- Modelling design or specifications rather than requirements or the Work. References to vocabulary about System:
  - references to database, Web page, clicks, taps, buttons, device, App, field, etc.
- Missing swimlanes or activities from secondary actors

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Process Model

- Process model represents a single function rather than a comprehensive use case
- Modelling design or specifications rather than requirements or the Work. References to vocabulary about System

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Scenarios

- Headers missing information
  - Triggers, preconditions not used correctly
- Alternatives should only be alternative desired behaviour
- Exceptions should only be desired recovery from bad events, inputs, situations
- Modelling design or specifications rather than requirements or the Work. References to vocabulary about System
- Some scenarios may need to be merged, to represent the scenarios of a comprehensive use case.
  - Important to get this right for Deliverable #6
  - Ask your TA if you did not get 10/10 for level of detail on the Use Case Model
- Inconsistent number of steps

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Atomic Requirements

- Just need one primary atomic requirement per sub scenario step
- Important to get this right for Deliverable #6

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Deliverable #6

100-Dollar Prioritization of comprehensive Use Cases
- with respect to importance/value
- by 4 stakeholders, 3 not on your team

Kano Prioritization of sub-scenarios (main scenario, alternatives, and exceptions) of the two highest-priority comprehensive Use Cases
- with respect to importance/value
- by 4 stakeholders, 3 not on your team

100-Dollar Prioritization of primary Atomic Requirements of the highest-priority sub-scenarios
- with respect to relative importance/value
- by 2 stakeholders not on your team

AHP Prioritization of primary Atomic Requirements of the highest-priority sub-scenarios
- with respect to relative difficulty to implement
- by all members your team

Plot relative value / relative difficulty of primary Atomic Requirements

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Specifications

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Objective

Want to articulate a software specification that meets the stakeholders' requirements

Refresh:

- Requirements - Conditions and capabilities that describe a *problem* – to be met by a *solution*, for the solution to be acceptable

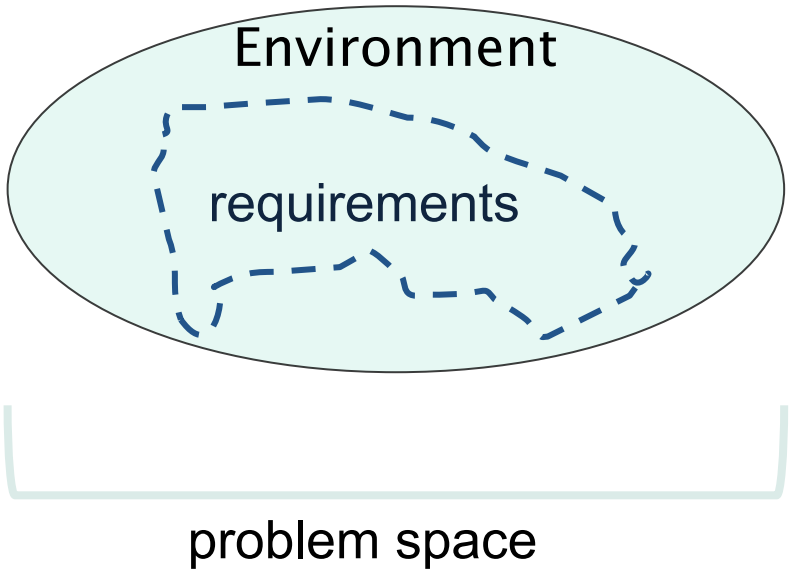
- Specification - A complete, precise, verifiable expression of requirements of a *software* or *system* solution.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Refresh: Requirements

A requirement is a condition or capability that must be achieved

- desired changes to the World
- expressed in terms of environmental phenomena

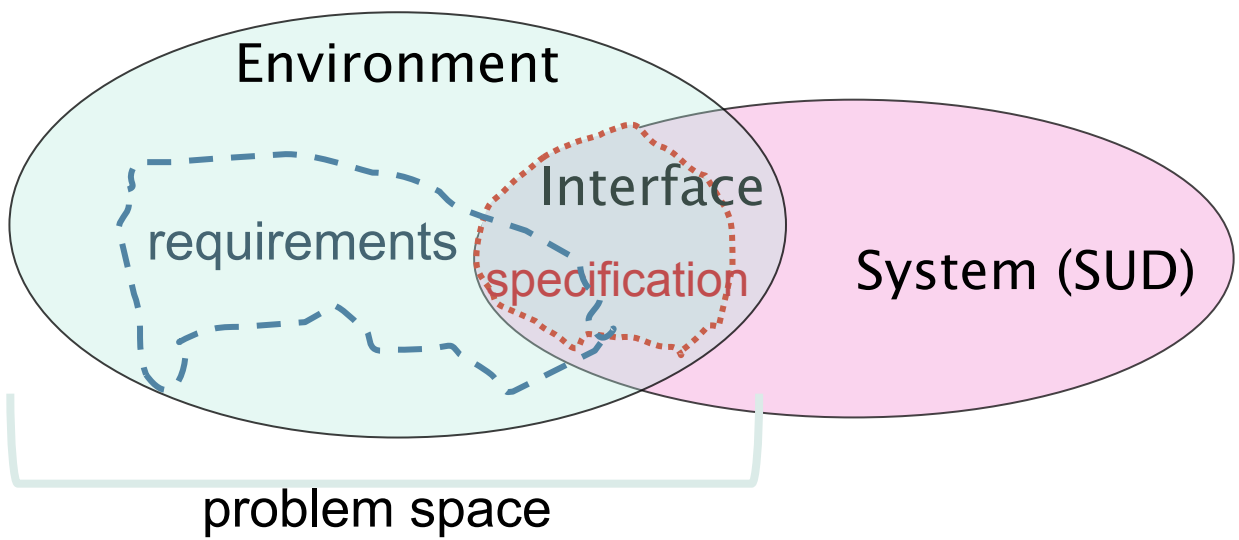


Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Refresh: Specification

A specification is a description of the proposed software system

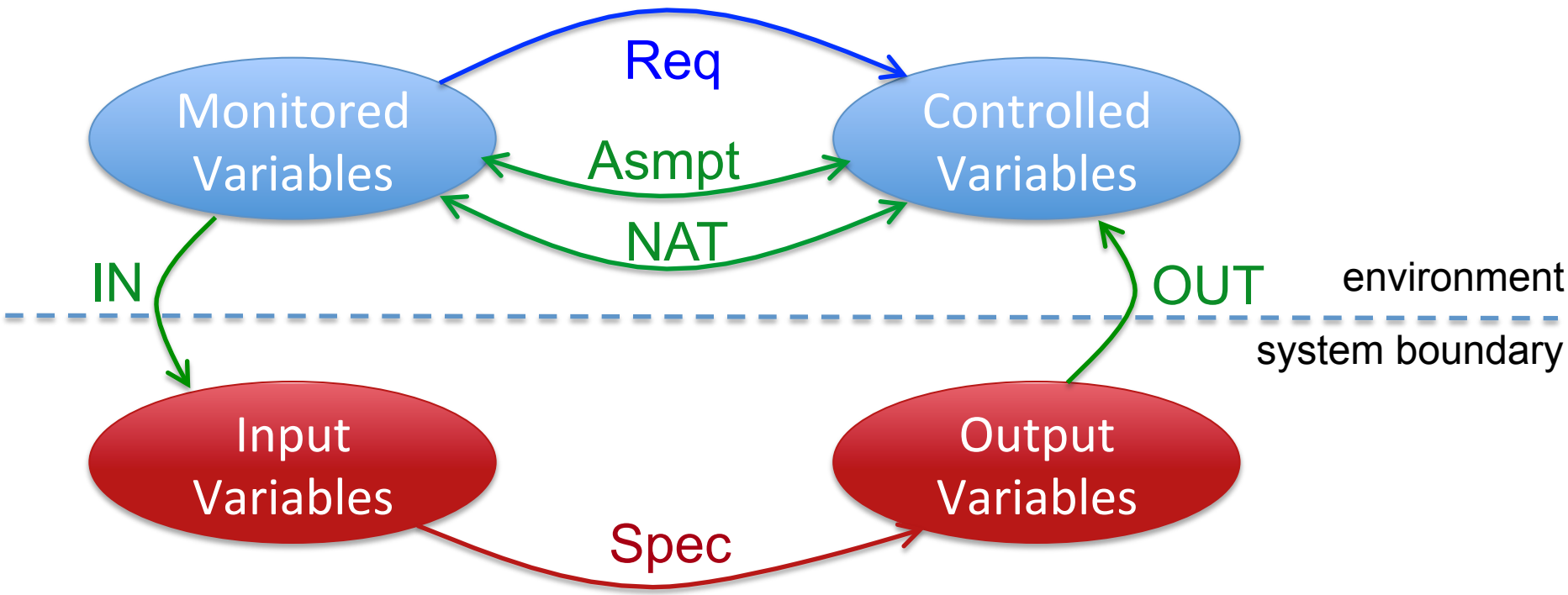
- system boundary is identified
  - with respect to what requirements it will implement
  - input data (from sensors, input devices)
  - output data or commands (to actuators, output devices)
- re-expresses requirements in terms of interface phenomena
- places no constraints on the design or implementation of the system giving the designer maximum freedom



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Requirements Vs. Specifications

Another way of seeing the distinction between requirements and specifications is by viewing them through the 4-variable model:



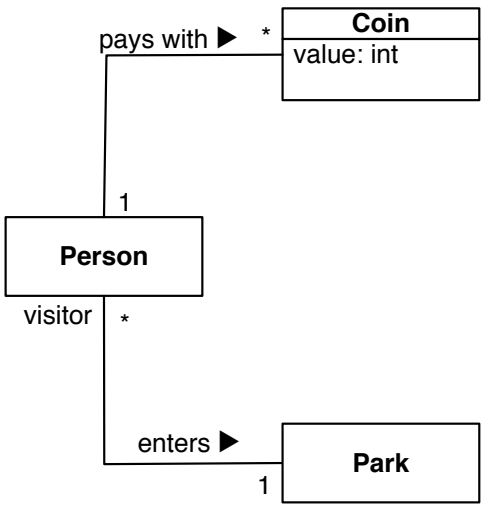
$$\text{Spec} \wedge \text{Dom} \models \text{Req}$$

$$\text{Dom} = \text{Asmpt} \wedge \text{NAT} \wedge \text{IN} \wedge \text{OUT}$$

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Park User Fees

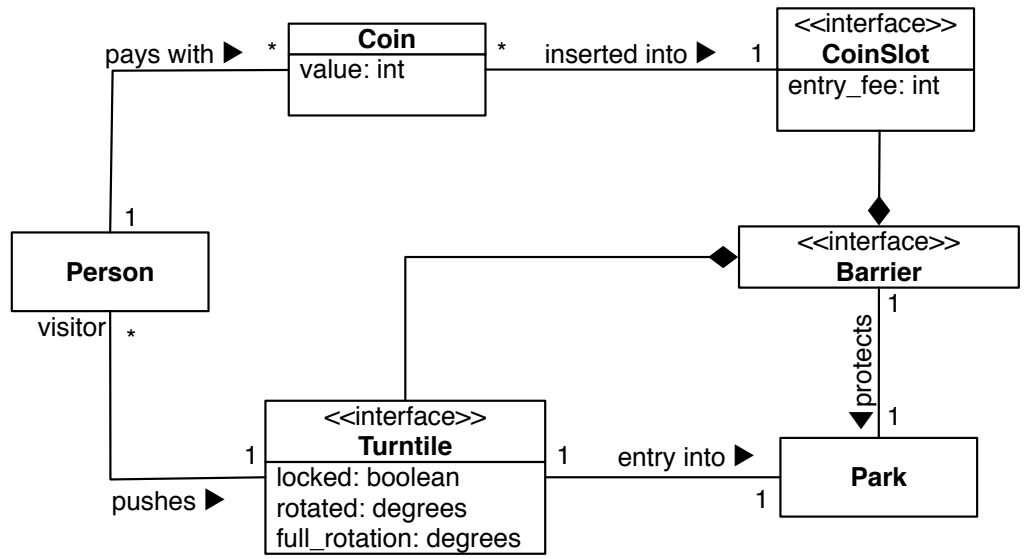
Requirements



Collect $1 from each visitor on entry to the park

Ensure that anyone who has paid can enter the park

Specifications



Collect $1 in the CoinSlot for every rotation of the Turnstile

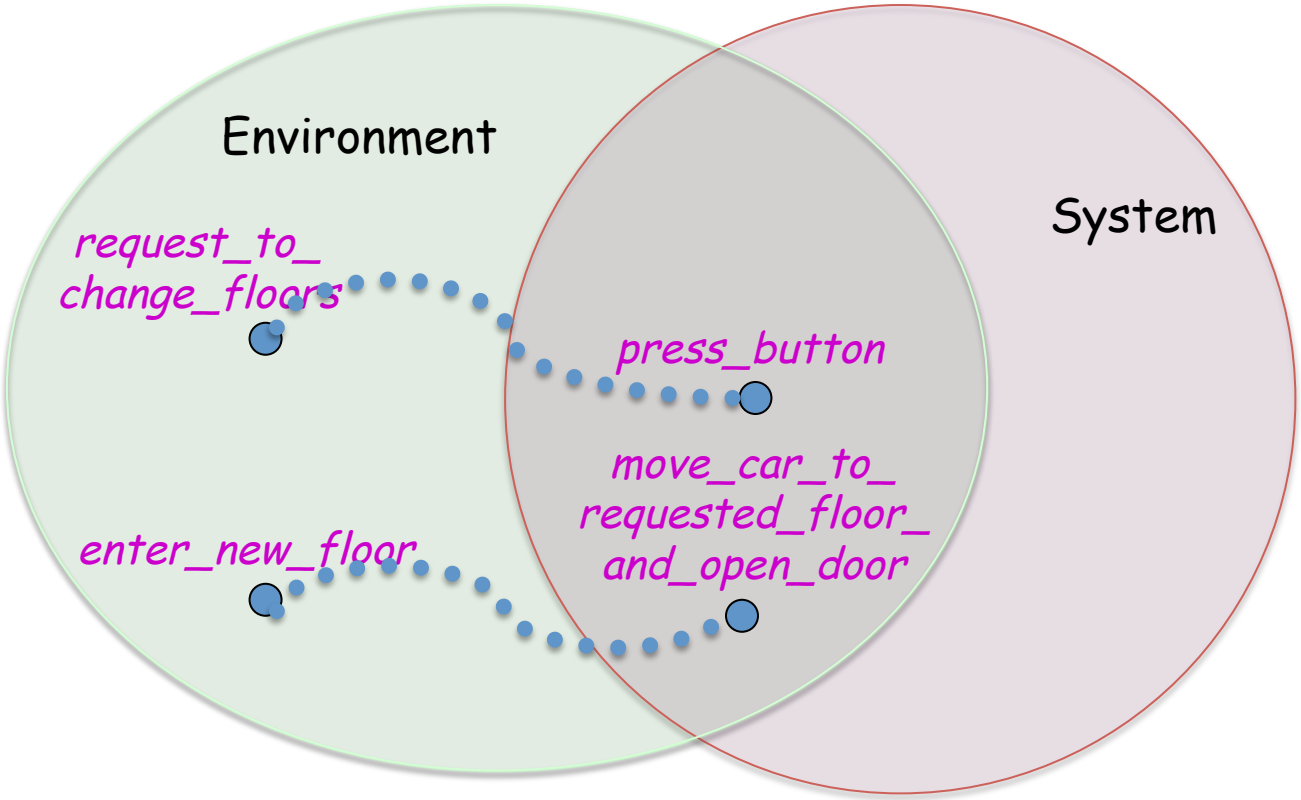
Whenever the CoinSlot receives $1, unlock the Turnstile

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Cyber-Physical Systems

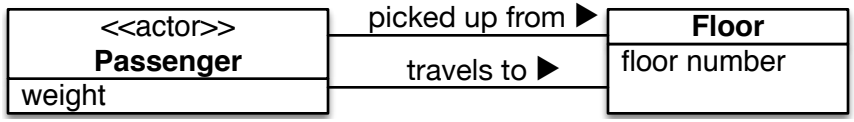
Interface entities are likely to be interface devices (sensors and actuators), sensor readings, and actuator commands

Example: Elevator



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Elevator



Req: Passenger is transported from her current floor to a designated floor

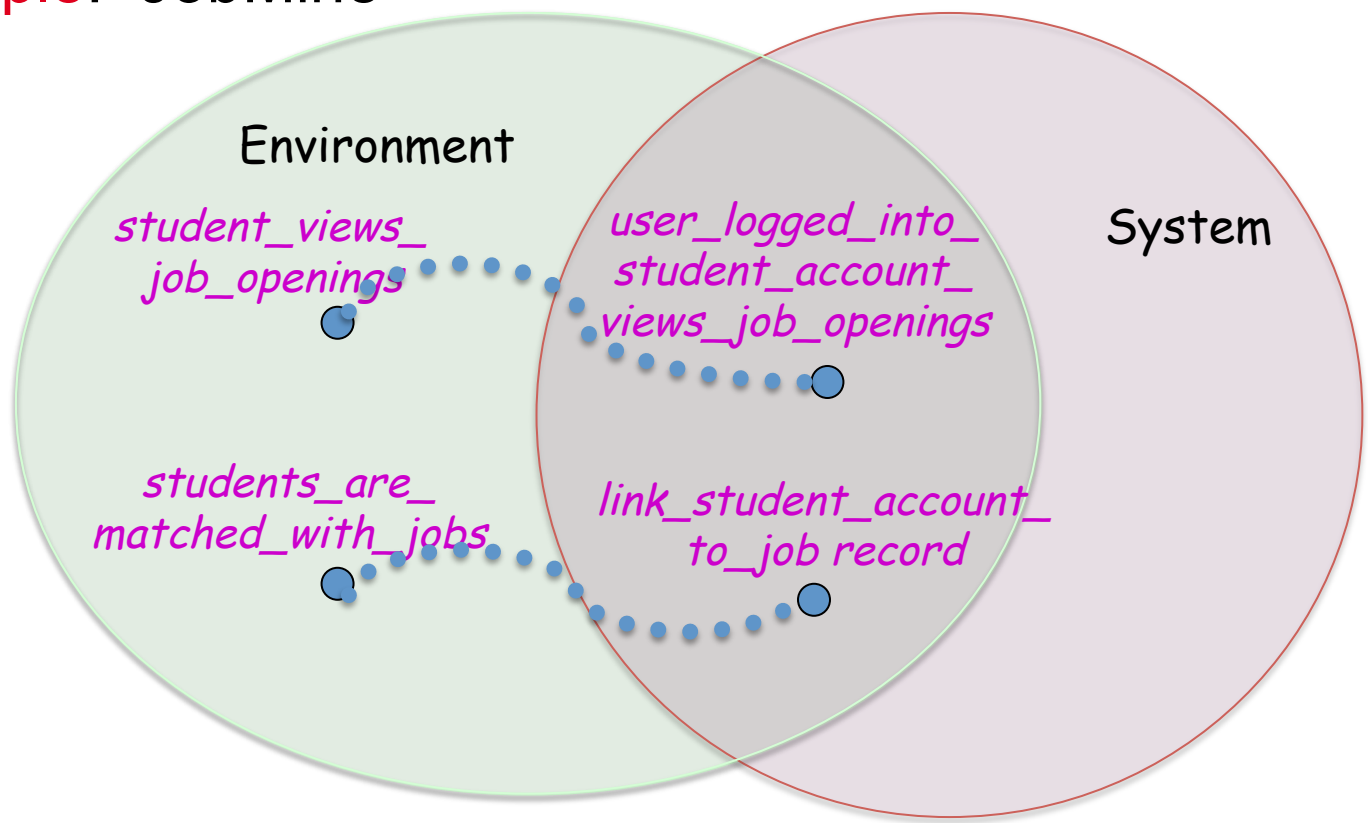
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Cyber Systems

Interface entities are likely to be cyber entities:

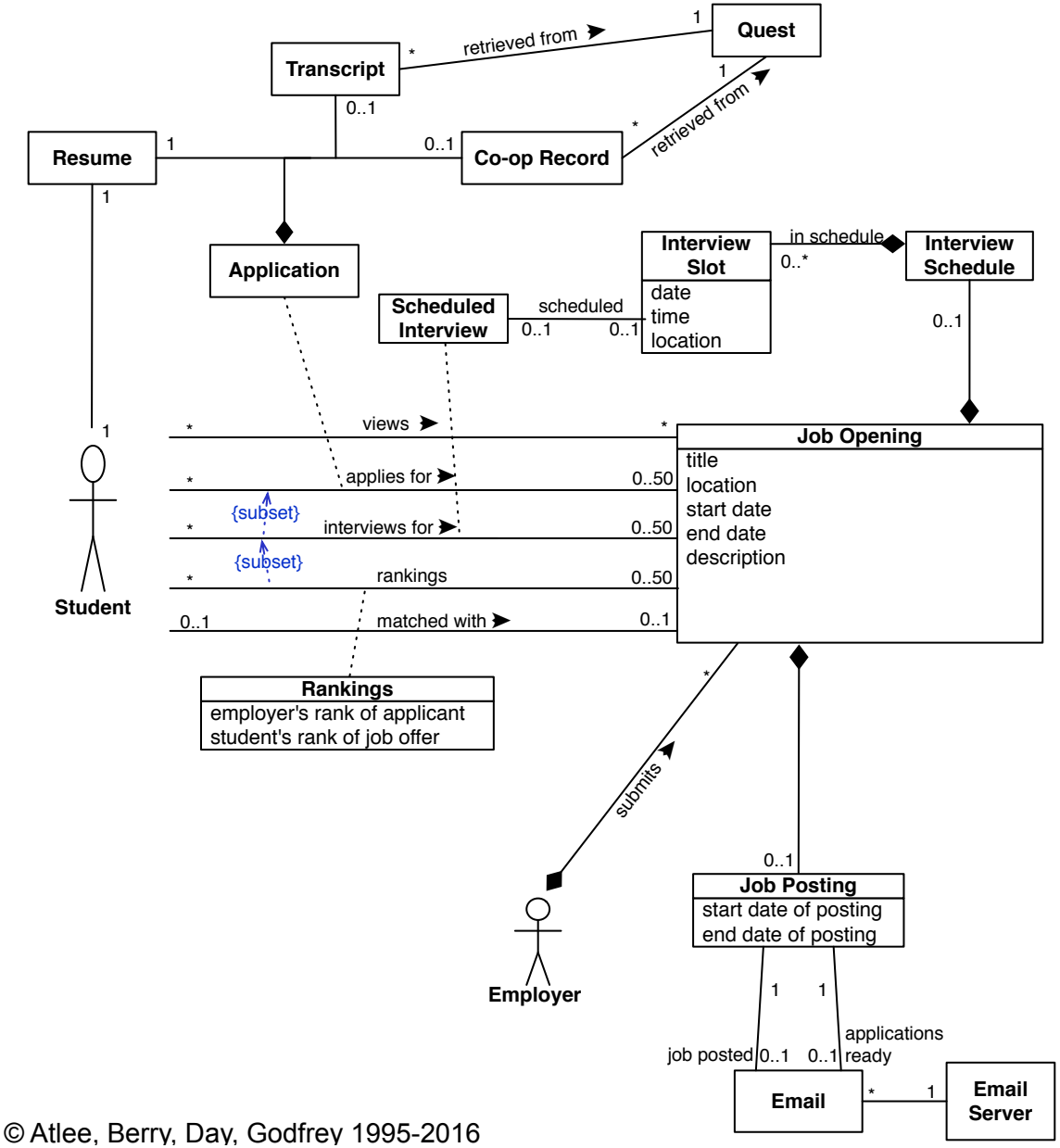
- Accounts, user ids, passwords, records, record ids, Web forms, keystrokes, mouse clicks, screen displays, popup messages

Example: JobMine



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: JobMine



R1: Student views Job Openings

R2: Product will match Students with Job Openings

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Requirements vs. Specifications

- Impact on Domain Model
- Impact on vocabulary used in models, scenarios, atomic requirements
- Spec, Dom $\models$ Req

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Deliverable #7

Updated Domain Models

- one for requirements descriptions (environment phenomena)
- one for specification descriptions (environment + interface pheno.)

Detailed Atomic Requirements

- conditions of satisfaction for N highest-priority primary atomic requirements
- expressed using environment phenomena

Detailed Atomic Specifications and Assumptions

- conditions of satisfaction for N highest-priority primary atomic requirements
- expressed using interface phenomena
- assumptions about environment phenomena, actors, input/output devices
    - needed to guarantee that a system meeting the specification will also meet the requirements

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Quality Requirements

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Quality Requirements

- Functional requirements describe what the software is supposed to do
  - What services or tasks the software should provide
  - Black box input/output behaviour

- Quality requirements describe (extra) constraints on what constitutes an acceptable software solution

  e.g., How fast system should respond

  Which deployment platforms should be supported

  How strong should security be

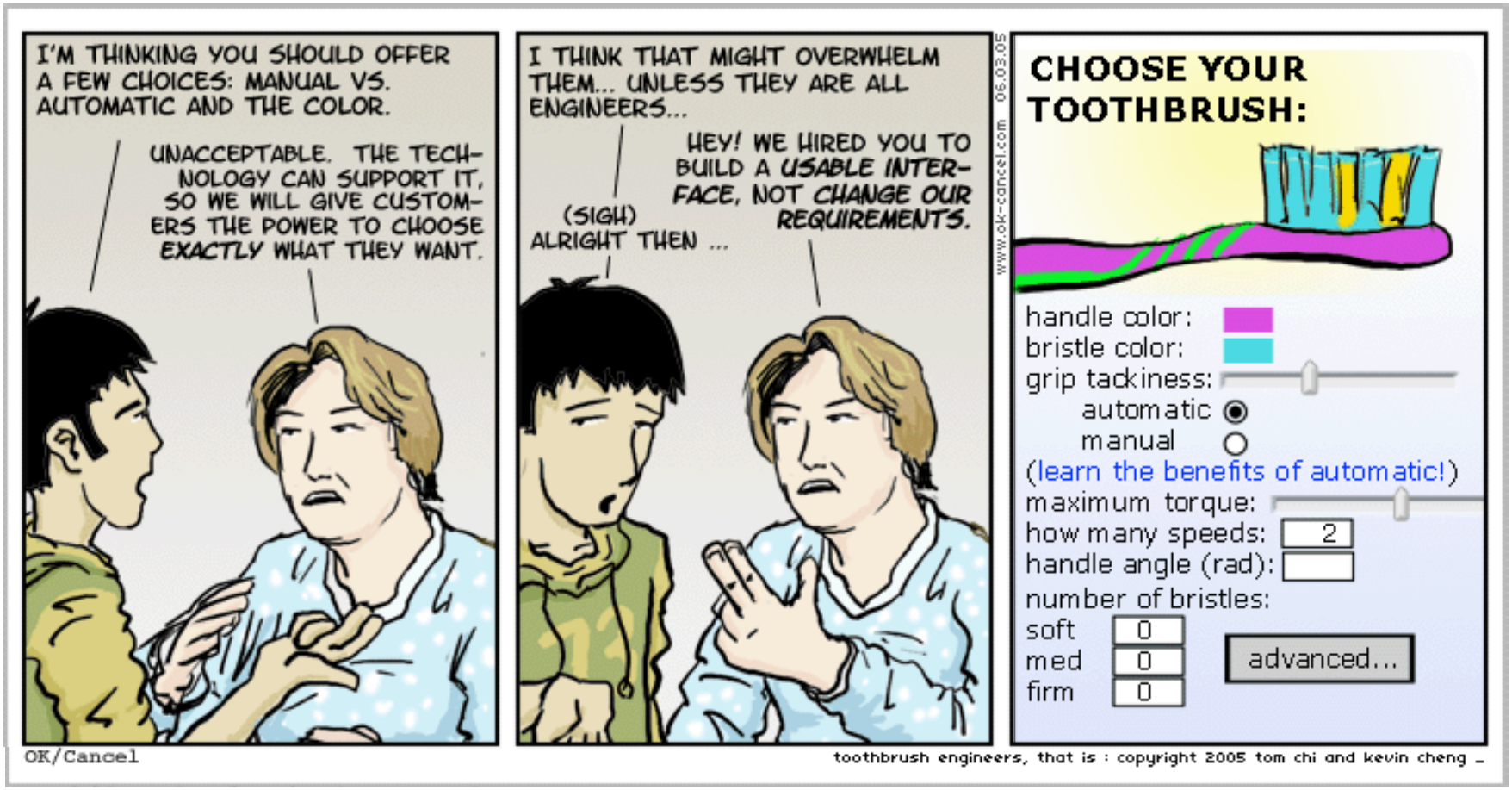
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Quality Requirements

- Performance
  - execution speed
  - response time
  - throughput
    - e.g., “up to 30 simultaneous calls”
- Reliability
  - fault-tolerant
  - mean-time to failure
  - data backups
- Robustness
  - tolerates invalid input
  - fault-tolerant
  - fail-safe / -secure
  - degrades gracefully under stress
- Adaptability
  - ease of adding new functionality
  - reusable in other environments
  - self-optimizing
  - self-healing
- Security
  - controlled access to system, data
  - isolation of data, programs
  - protect against theft, vandalism
- Usability
  - how easy to learn / use
  - user productivity
- Scalability
  - workload
  - number of users
  - size of data sets
  - peak use
- Efficiency (capacity)
  - user productivity
  - utilization of resources
- Accuracy / precision
  - tolerance of computation errors
  - precision of computation results

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Useability



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Other Nonfunctional Requirements

Design constraints

- interfaces to other systems
- COTS components
- programming language

Operating Constraints

- location
- size, power consumption
- temperature, humidity
- operating costs
- accessibility (for maintenance)

Product-family requirements

- modifiability
- portability
- reusability
- UI

Process Requirements

- Resources
  - personnel development
  - costs
  - development schedule
- Documentation
  - audience
  - conventions
  - readability
- Complexity (of code)
  - comments / KLOC
  - coupling / cohesion
  - cyclomatic complexity
  - use of multiple inherit. overloading, templates
- Standards compliance

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

“Motherhood” requirements

Expressions such as “reliable”, “user-friendly”, and “maintainable” are motherhood requirements.

- No one would explicitly ask their opposite

e.g., *slow, unreliable, user-hostile, unmaintainable*, …

(Virtually) every software system must have attributes such as “reliable”, “user-friendly”, and “maintainable”; what differs from product to product is:

- the degree to which each attribute is required, and
- the relative importance of one attribute over another.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Fit Criteria

"Anyone can build a bridge. It takes an engineer to build a bridge that barely stands."

[unknown source]

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Fit criteria

A fit criterion quantifies the extent to which a quality requirement must be met.

Example:

- *75% of users shall judge the system to be as usable as the existing system*
- *After training, 90% of users shall be able to process a new account within 4 minutes*
- *A module will encapsulate the data representation of at most one data type*
- *Computation errors shall be fixed within 3 weeks of being reported*

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Richer Fit Criteria

Requirement	Outstanding	Target	Minimum
Response time	0.1s	0.5s	1s
CPU utilization	20%	25%	30%
Usability	40 tasks/hr	30 tasks/hr	20 tasks/hr

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Measurements

What gets measured gets done!

- Therefore, unless a quantified requirement is unrealistic, it will probably be met.
- There is a danger of focusing on what is measurable, and not on the true requirement
  - e.g., industry benchmarks (sometimes)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

When you can't test before delivery

- Fit criteria that cannot be evaluated before the final product is delivered are harder to assess. For example:
  - *The system shall not be unavailable for more than a total of 3 minutes each year*
  - *The mean-time-to-failure shall be no less than 1 year*

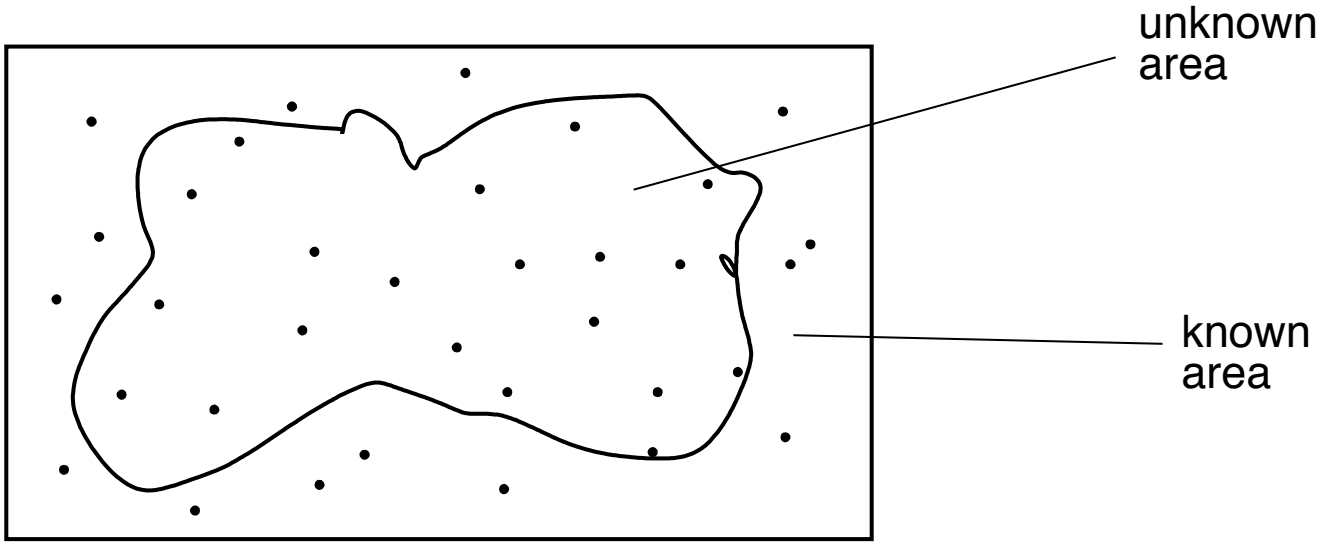
- Possible approaches:
  - Measure the attributes of a prototype.
  - Measure secondary indicators

    e.g., number of user errors to assess usability
  - Estimate a system's quality attributes
  - Deliver system and pay penalty if requirements are not met

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Monte Carlo techniques

Monte Carlo techniques: estimate an unknown quantity using a known quantity.



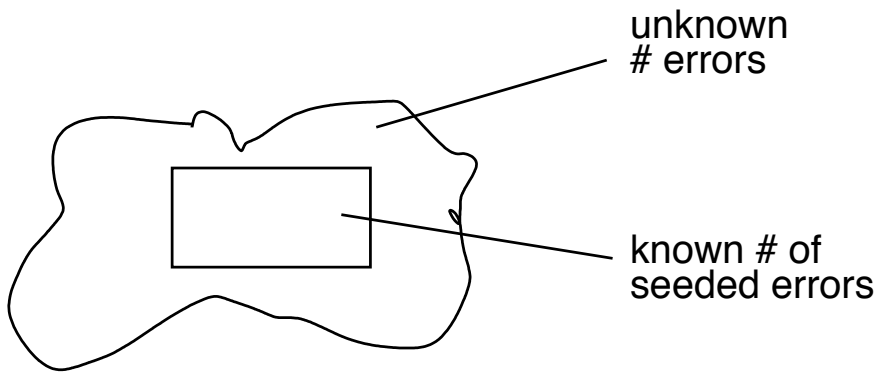
$$\frac{\text{Number of points in shape}}{\text{Total number of points}} \approx \frac{\text{Area of Shape}}{\text{Known Area of Rectangle}}$$

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Monte Carlo techniques

We can use Monte Carlo techniques to estimate number of bugs remaining in a program (reliability).

- Plant a known number of errors into the program, which the testing team does not know about.
- Then compare the number of seeded errors the team detects with the total number of errors it detects, to arrive at an estimate of the total number of bugs in the program.



$$\frac{\text{\# detected seeded errors}}{\text{\# seeded errors}} \approx \frac{\text{\# detected errors}}{\text{\# errors in the program}}$$

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Paper Airplane Exercise

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Prioritizing Quality Requirements

Many quality requirements conflict with one another:

- maintainability vs. robustness
  - simple design vs. design that monitors run-time / error recovery
- performance vs. security
- performance vs. reuse
- performance vs. portability
- robustness vs. testability

Also, some quality requirements conflicts with functional requirements

- performance vs. particular features (e.g., unlimited undo)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Typical Conflicts

	Availability	Efficiency	Installability	Integrity	Interoperability	Modifiability	Performance	Portability	Reliability	Reusability	Robustness	Safety	Scalability	Security	Usability	Verifiability
Availability									+		+					
Efficiency	+				−	−	+	−			−		+		−	
Installability	+								+					+		
Integrity			−		−		−			−		+		+	−	−
Interoperability	+		−	−			−	+	+		+	−		−		
Modifiability	+		−				−		+	+			+			+
Performance		+			−	−		−			−		−		−	
Portability		−			+	−	−			+				−	−	+
Reliability	+	−		+		+	−				+	+		+	+	+
Reusability		−		−	+	+	−	+						−		+
Robustness	+	−	+	+	+		−		+			+	+	+	+	
Safety		−		+	+		−				+			+	−	−
Scalability	+	+		+			+	+	+		+					
Security	+			+	+		−	−	+		+	+			−	−
Usability		−	+				−	−	+		+	+				−
Verifiability	+		+	+		+			+	+	+	+		+	+	

FIGURE 14-2 Positive and negative relationships among selected quality attributes.

Karl Wiegers, *Software Requirements*, (MS Press)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Quality requirements affect how a system accomplishes its functional goals

- Quality requirements are highly important to user experience
- Multiple types to consider
- Need to elicit and specify preferences
- Need to elicit and specify fit criteria

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Deliverable #8

- Identify 6 quality requirements that your team believes are important to your project
  - State what the quality requirement means in the context of your project
  - Can have multiple requirements related to the same quality attribute, but must involve at least 4 quality attributes
- Use 100-dollar technique to prioritize quality requirements
  - Involve at least 2 concrete stakeholders, 1 not from your team
- Elicit fit criteria for top 3 quality requirements
  - Involve at least 2 concrete stakeholders, 1 not from your team

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Risk Management

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk

A risk is an uncertain factor whose occurrence may result in some loss of satisfaction of some corresponding objective.

[van Lamsveerde, p. 3.6]

- has a likelihood to occur
- has consequences
- product-related risks
- process-related risks

Project Risk Manager

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

No Risk, No Reward

Trying to completely eliminate risk from your software project is unrealistic and can be prohibitively expensive.

"Gain is commensurate with risk."

M.K. Soni

"He who doesn't risk never gets to drink champagne."

Russian Proverb

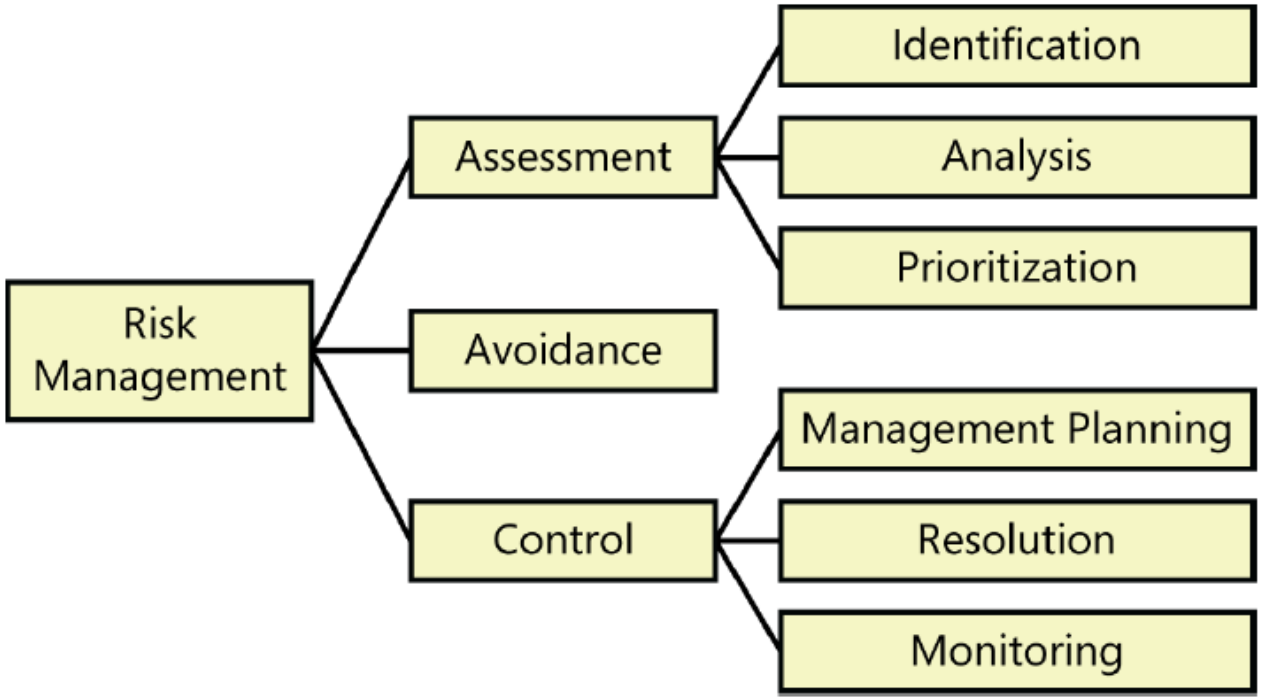
"Don't be afraid to take a big step when one is indicated. You can't cross a chasm in two small steps."

David Lloyd George

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk Management

Risk Management attempts to manage the degree to which a project is exposed to risks of quality, delay, or failure.



K. Wiegers and J. Beatty, Software Requirements 3ed, Microsoft Press, 2013.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk Management

"Risk is like fire: If controlled it will help you; if uncontrolled it will rise up and destroy you."

Theodore Roosevelt

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Defect Detection and Prevention (DDP)

DDP is a process developed by NASA for systematic risk assessment and mitigation.

DDP Process:

1. Identify most critical requirements
2. Identify potential risks
3. Estimate the impact of each risk on each requirement
4. Identify possible countermeasures
5. Identify the most effective countermeasures

Result of DDP: Optimized collection of mitigating actions that may be applied to project

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk Consequence Table

DDP Process:

1. Identify most critical requirements, and their relative importance
2. Identify potential risks, and their likelihood
3. Estimate the impact of each risk on each requirement
4. Identify possible countermeasures
5. Identify the most effective countermeasures

Goal:

- To develop a prioritized set of risks to be addressed
- Perhaps to identify which requirements are the most “risk driving”

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk Consequence Table

		Risks (Failure Modes)				
Requirements	**Weight (req)**	**Requirements** (incomplete, incorrect, ambiguous, changing)	**Project Management** (estimations, project, team management)	**Technical** (complex problem, lack of experience with technology)	**Dependencies** (on adjacent systems, components, other people)	**Loss of Objective**
Likelihood (risk)		**0.5**	**0.8**	**0.7**	**0.2**	
Creating a product that users would like	**0.7**	0.9	0.3	0.9	0.5	**0.7**
Completing the product on time	**1.0**	0.8	1	0.9	0.3	**1.96**
Risk Criticality		**0.715**	**0.968**	**1.071**	**0.13**	

Impact(risk, req) = estimate of loss of requirement

0 = no loss

1 = total loss

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Loss of Objectives

		Risks (Failure Modes)				
Requirements	**Weight (req)**	**Requirements** (incomplete, incorrect, ambiguous, changing)	**Project Management** (estimations, project, team management)	**Technical** (complex problem, lack of experience with technology)	**Dependencies** (on adjacent systems, components, other people)	**Loss of Objective**
Likelihood (risk)		**0.5**	**0.8**	**0.7**	**0.2**	
Creating a product that users would like	**0.7**	0.9	0.3	0.9	0.5	**0.7**
Completing the product on time	**1.0**	0.8	1	0.9	0.3	**1.96**
Risk Criticality		**0.715**	**0.968**	**1.071**	**0.13**	

$$\text{Loss(req)} = \text{Weight(req)} \times \sum_{\text{risk}} \left(\text{Impact(req,risk)} * \text{Likelihood(risk)}\right)$$

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk-Driving Requirements

		Risks (Failure Modes)				
Requirements	**Weight (req)**	**Requirements** (incomplete, incorrect, ambiguous, changing)	**Project Management** (estimations, project, team management)	**Technical** (complex problem, lack of experience with technology)	**Dependencies** (on adjacent systems, components, other people)	**Loss of Objective**
Likelihood (risk)		**0.5**	**0.8**	**0.7**	**0.2**	
Creating a product that users would like	**0.7**	0.9	0.3	0.9	0.5	**0.7**
Completing the product on time	**1.0**	0.8	1	0.9	0.3	**1.96**
Risk Criticality		**0.715**	**0.968**	**1.071**	**0.13**	

Risk-driving requirement

Risk-driving requirements are the requirements that are most at risk of being achieved.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk Criticalities

		Risks (Failure Modes)				
Requirements	**Weight (req)**	**Requirements** (incomplete, incorrect, ambiguous, changing)	**Project Management** (estimations, project, team management)	**Technical** (complex problem, lack of experience with technology)	**Dependencies** (on adjacent systems, components, other people)	**Loss of Objective**
Likelihood (risk)		**0.5**	**0.8**	**0.7**	**0.2**	
Creating a product that users would like	**0.7**	0.9	0.3	0.9	0.5	**0.7**
Completing the product on time	**1.0**	0.8	1	0.9	0.3	**1.96**
Risk Criticality		**0.715**	**0.968**	**1.071**	**0.13**	

$$\text{Criticality(risk)} = \text{Likelihood(risk)} \times \sum_{\text{req}} \left(\text{Impact(req,risk)} * \text{Weight(req)}\right)$$

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Tall Poles

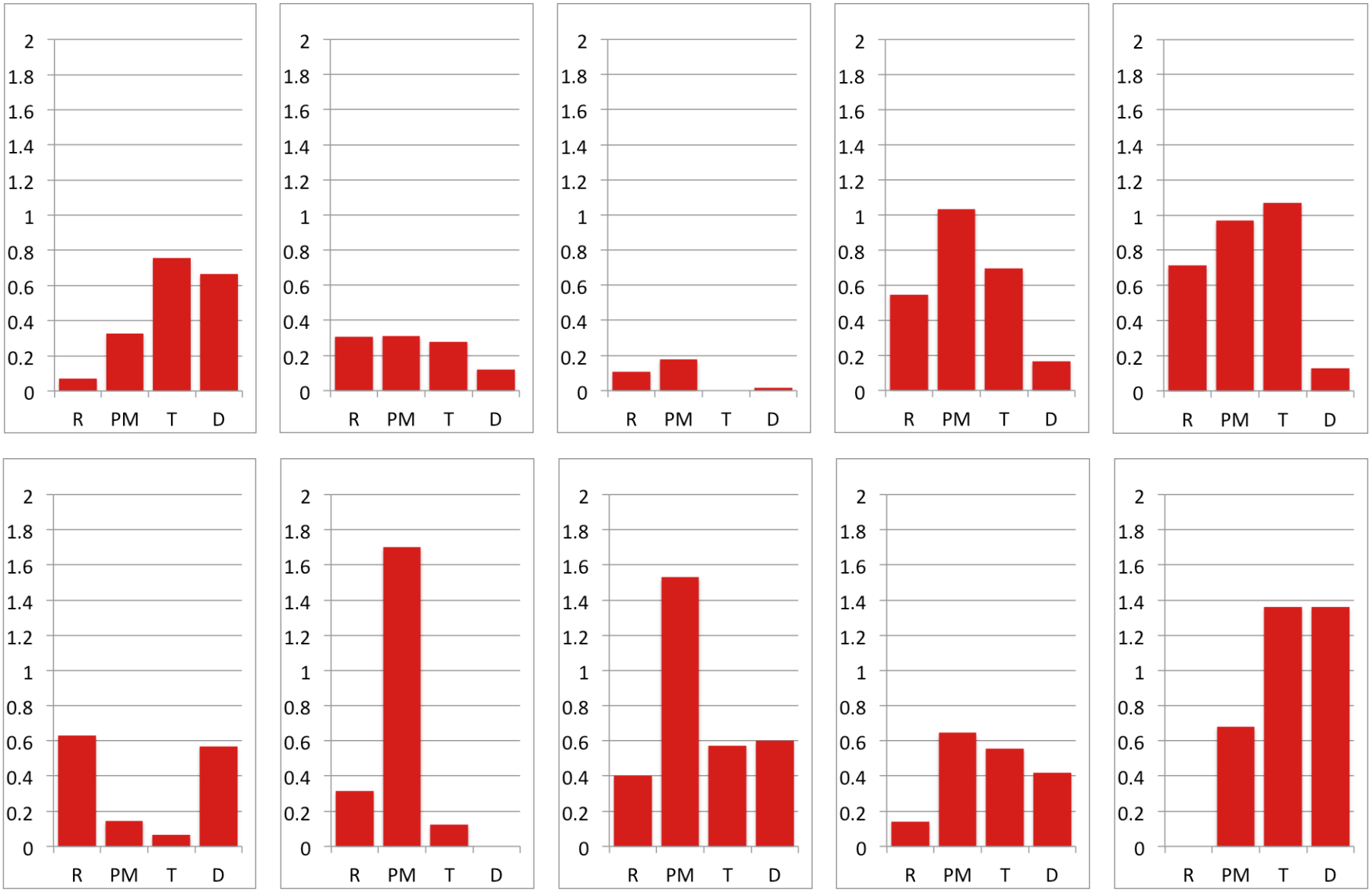
		Risks (Failure Modes)				
Requirements	**Weight (req)**	**Requirements** (incomplete, incorrect, ambiguous, changing)	**Project Management** (estimations, project, team management)	**Technical** (complex problem, lack of experience with technology)	**Dependencies** (on adjacent systems, components, other people)	**Loss of Objective**
Likelihood (risk)		**0.5**	**0.8**	**0.7**	**0.2**	
Creating a product that users would like	**0.7**	0.9	0.3	0.9	0.5	**0.7**
Completing the product on time	**1.0**	0.8	1	0.9	0.3	**1.96**
Risk Criticality		**0.715**	**0.968**	**1.071**	**0.13**	

Tall Poles

Tall Poles are the most critical risks, having the most severe consequences

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Teams' Assessments of Risks



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk Management

"One thing that makes is possible to be an optimist is if you have a contingency plan for when all hell breaks loose."

Randy Pausch

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk Countermeasures Table

DDP Process:

1. Identify most critical requirements
2. Identify potential risks, and their likelihood
3. Estimate the impact of each risk on each requirement
4. Identify possible countermeasures, and their effectiveness in reducing risk
5. Identify the most effective countermeasures

Goal:

- Identify options for preventing or detecting failure modes
  - Preventative measures, Analyses, Process controls, Tests, Mitigations
- Perhaps to identify the most effective countermeasures

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk Countermeasures Table

	Risks (Failure Modes)				
Countermeasures	**Requirements** (incomplete, incorrect, ambiguous, changing)	**Project Management** (estimations, project, team management)	**Technical** (complex problem, lack of experience with technology)	**Dependencies** (on adjacent systems, components, other people)	**Overall single effect of countermeasure**
Criticality (risk)	**0.715**	**0.968**	**1.071**	**0.13**	
Collaborative elicitation process with extensive user involvement; modelling; mock-ups	0.5	0.3	0	0.1	**0.6479**
Continually estimate costs; use shorter development iterations	0.25	0.7	0.2	0	**1.09655**
Prototype novel or risky requirements; plan time for learning and experimentation	0.25	0.25	0.5	0	**1.02125**
Investigate suppliers; monitor their progress; develop backup plans	0	0	0.1	0.5	**0.1201**
Combined Risk Reduction	**0.71875**	**0.8425**	**0.64**	**0.55**	

Effect(cm, risk) = estimate of reduction of risk

0 = no reduction

1 = risk eliminated

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Overall Effect of Countermeasures

	Risks (Failure Modes)				
Countermeasures	**Requirements** (incomplete, incorrect, ambiguous, changing)	**Project Management** (estimations, project, team management)	**Technical** (complex problem, lack of experience with technology)	**Dependencies** (on adjacent systems, components, other people)	**Overall single effect of countermeasure**
Criticality (risk)	**0.715**	**0.968**	**1.071**	**0.13**	
Collaborative elicitation process with extensive user involvement; modelling; mock-ups	0.5	0.3	0	0.1	**0.6479**
Continually estimate costs; use shorter development iterations	0.25	0.7	0.2	0	**1.09655**
Prototype novel or risky requirements; plan time for learning and experimentation	0.25	0.25	0.5	0	**1.02125**
Investigate suppliers; monitor their progress; develop backup plans	0	0	0.1	0.5	**0.1201**
Combined Risk Reduction	**0.71875**	**0.8425**	**0.64**	**0.55**	

$$\text{OverallEffect}(cm) = \sum_{risk} \left(\text{Reduction}(cm,risk) * \text{Criticality}(risk)\right)$$

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Combined Risk Reduction

	Risks (Failure Modes)				
Countermeasures	**Requirements** (incomplete, incorrect, ambiguous, changing)	**Project Management** (estimations, project, team management)	**Technical** (complex problem, lack of experience with technology)	**Dependencies** (on adjacent systems, components, other people)	**Overall single effect of countermeasure**
Criticality (risk)	**0.715**	**0.968**	**1.071**	**0.13**	
Collaborative elicitation process with extensive user involvement; modelling; mock-ups	0.5	0.3	0	0.1	**0.6479**
Continually estimate costs; use shorter development iterations	0.25	0.7	0.2	0	**1.09655**
Prototype novel or risky requirements; plan time for learning and experimentation	0.25	0.25	0.5	0	**1.02125**
Investigate suppliers; monitor their progress; develop backup plans	0	0	0.1	0.5	**0.1201**
Combined Risk Reduction	**0.71875**	**0.8425**	**0.64**	**0.55**	

$$\text{CombinedReduction(risk)} = 1 - \prod_{cm} \left(1 - \text{Reduction(cm,risk)}\right)$$

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

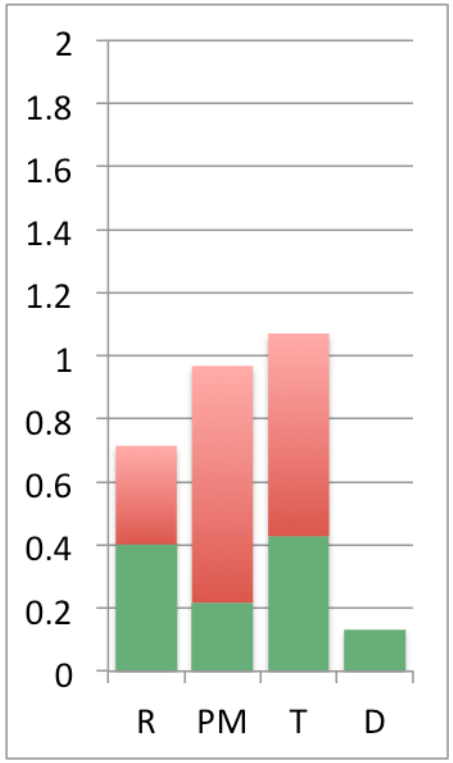
Apply Optimal Countermeasures

		Risks (Failure Modes)				
Requirements	**Weight (req)**	**Requirements** (incomplete, incorrect, ambiguous, changing)	**Project Management** (estimations, project, team management)	**Technical** (complex problem, lack of experience with technology)	**Dependencies** (on adjacent systems, components, other people)	**Loss of Objective**
Mitigated Likelihood (risk)		**0.28125**	**0.18**	**0.28**	**0.2**	
Creating a product that users would like	**0.7**	0.9	0.3	0.9	0.5	**0.3437875**
Completing the product on time	**1.0**	0.8	1	0.9	0.3	**0.745**
Mitigated Risk Criticality		**0.4021875**	**0.2178**	**0.4284**	**0.13**	

$$\text{MitigatedLikelihood(risk)} = \text{Likelihood(risk)} * \prod \left(1 - \text{Reduction(cm,risk)}\right)$$

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Apply Optimal Countermeasures



Red bars show the risk levels before mitigation.

Green bars show the risk levels after mitigation.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Meeting Scheduler

- A meeting initiator informs potential participants about the need for a meeting and specifies a date range within which the meeting should take place, asking them to return their scheduling constraints
- Constraints are expressed as two sets:
  - one **exclusion set (dates when a participant cannot attend)**
  - one **preference set (dates when a participant prefers to attend)**
- Initiator also asks for specific requirements of meeting room
- All correspondence with participants is via email
- The meeting should be scheduled within the stated date range and not be in any exclusion sets. The date should also belong to as many preference sets as possible, especially of the “important” participants.
- A new schedule cycle is required in case of date or room conflict.
- Conflicts can be resolved in several ways: the initiator may extend the date range, some participants may remove dates from their exclusion set, or some participants may decline the invitation to attend the meeting.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk Consequence Table

		Risks (Failure Modes)					
Requirements	**Weight (req)**	Participant does not read emails	Participant does not reply to requests	Room with required equipment is not available	System response is too close to meeting	Important participant has last minute change	**Loss of objective**
Likelihood (risk)		**0.4**	**0.3**	**0.1**	**0.3**	**0.5**	
Time taken to schedule meetings reduced	**0.5**	0.6	0.8	0.2	0.7	0.2	**0.405**
Send out notification asap when time and place are found	**0.4**	0	0.8	0	1	0.2	**0.256**
Participant average attendance increased	**0.3**	0.8	0.8	0	0.8	0.5	**0.315**
Schedule conflicts reduced	**0.6**	0.2	1	0	0	0.7	**0.438**
Risk criticality		**0.264**	**0.468**	**0.01**	**0.297**	**0.375**	

Impact(risk, req) = estimate of loss of requirement

0 = no loss

1 = total loss

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk-Driving Requirements

		Risks (Failure Modes)					
Requirements	**Weight (req)**	Participant does not read emails	Participant does not reply to requests	Room with required equipment is not available	System response is too close to meeting	Important participant has last minute change	**Loss of objective**
Likelihood (risk)		**0.4**	**0.3**	**0.1**	**0.3**	**0.5**	
Time taken to schedule meetings reduced	**0.5**	0.6	0.8	0.2	0.7	0.2	**0.405**
Send out notification asap when time and place are found	**0.4**	0		0	1	0.2	**0.256**
Participant average attendance increased	**0.3**	0.8	0.8	0	0.8	0.5	**0.315**
Schedule conflicts reduced	**0.6**	0.2	1	0	0	0.7	**0.438**
Risk criticality		**0.264**	**0.468**	**0.01**	**0.297**	**0.375**	

Risk driving

Risk-driving requirements are the requirements that are most at risk of not being achieved

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Tall Poles

Tall Poles

		Risks (Failure Modes)					
Requirements	**Weight (req)**	Participant does not read emails	Participant does not reply to requests	Room with required equipment is not available	System response is too close to meeting	Important participant has last minute change	**Loss of objective**
Likelihood (risk)		**0.4**	**0.3**	**0.1**	**0.3**	**0.5**	
Time taken to schedule meetings reduced	**0.5**	0.6	0.8	0.2	0.7	0.2	**0.405**
Send out notification asap when time and place are found	**0.4**	0		0	1	0.2	**0.256**
Participant average attendance increased	**0.3**	0.8	0.8	0	0.8	0.5	**0.315**
Schedule conflicts reduced	**0.6**	0.2	1	0	0	0.7	**0.438**
Risk criticality		**0.264**	**0.468**	**0.01**	**0.297**	**0.375**	

Tall Poles are the most critical risks, having the most severe consequences

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Risk Countermeasures Table

	Risks (Failure Modes)					
Countermeasures	Participant does not read emails	Participant does not reply to requests	Room with required equipment is not available	System response is too close to meeting	Important participant has last minute change	**Overall single effect of countermeasure**
Criticality (risk)	**0.264**	**0.468**	**0.01**	**0.297**	**0.375**	
Email reminder sent	0.7	0.7	0	0.1	0	**0.542**
Change the meeting, increase time range	0.2	0.2	0	0.1	0	**0.176**
System has access to personal e-agendas	0.3	0.2	0.1	0.2	0.3	**0.346**
Change the meeting, fewer constraints (equipment)	0	0	0.9	0	0	**.009**
Cancel a meeting and send email confirmation	0.8	0.8	1	0.7	0.9	**1.141**
Combined risk reduction	**0.966**	**0.962**	**1**	**0.806**	**0.93**	

Effect(cm, risk) = estimate of reduction of risk

0 = no reduction

1 = risk eliminated

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Combined Risk Reduction

	Risks (Failure Modes)					
Countermeasures	Participant does not read emails	Participant does not reply to requests	Room with required equipment is not available	System response is too close to meeting	Important participant has last minute change	**Overall single effect of countermeasure**
Criticality (risk)	**0.264**	**0.468**	**0.01**	**0.297**	**0.375**	
Email reminder sent	0.7	0.7	0	0.1	0	**0.542**
Change the meeting, increase time range	0.2	0.2	0	0.1	0	**0.176**
System has access to personal e-agendas	0.3	0.2	0.1	0.2	0.3	**0.346**
Change the meeting, fewer constraints (equipment)	0	0	0.9	0	0	**.009**
Cancel a meeting and send email confirmation	0.8	0.8	1	0.7	0.9	**1.141**
Combined risk reduction	**0.966**	**0.962**	**1**	**0.806**	**0.93**	

$$\text{CombinedReduction(risk)} = 1 - \prod_{cm} (1 - \text{Reduction(cm, risk)})$$

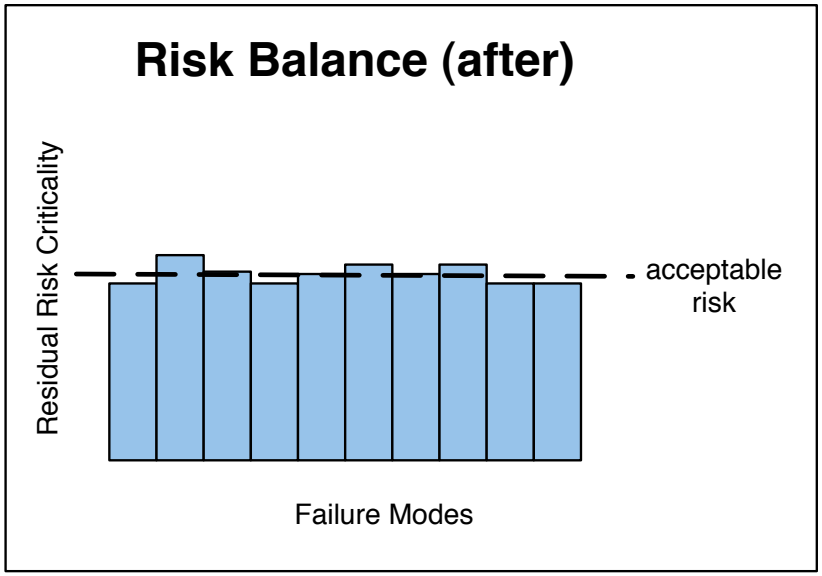
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Optimizing the Residual Risk

Goal: Select the optimal combinations of countermeasures

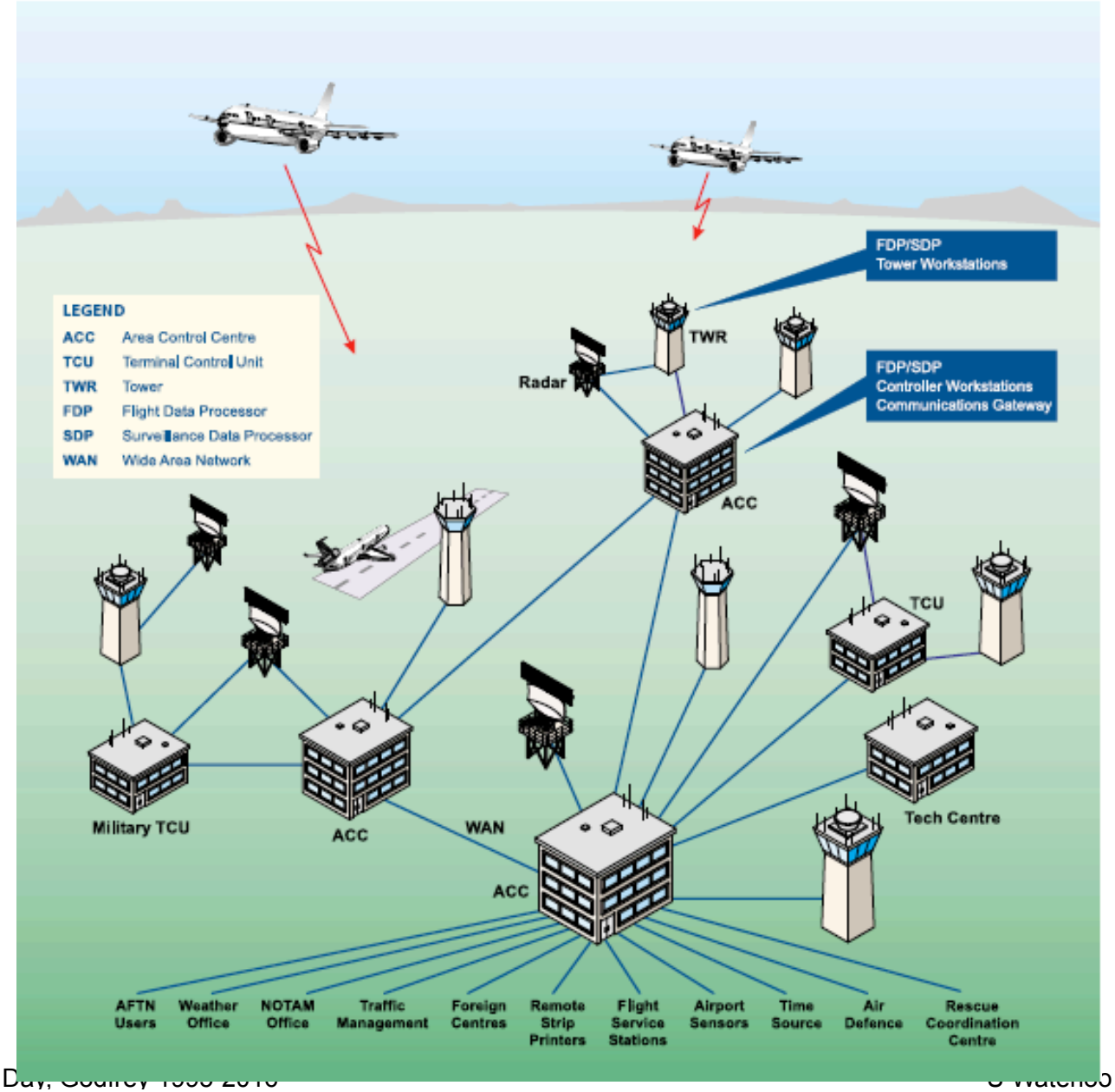
- based on their joint effectiveness in reducing risk
- based on their associated cost of implementation

Result: Optimal (or near optimal) balance of risk exposure



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Canadian Automated Air Traffic Control System



Case Study: CAATS

- Goals of project were to provide
  - Paperless, integrated flight data to controllers
  - Automatic conflict detection and prediction
  - Automatic proposals for conflict resolution
- Project completed
  - 64 months later than originally planned
  - With a cost >$123 million more than originally planned
  - With reduced functionality
    - excluded hardware and some commercial software
    - no automatic proposal of conflict resolution
    - to be installed at 23 control towers instead of 60
    - no automatic management of military altitude reservation
    - reduced traffic flow management features
    - transition to CAATS to be gradual rather than synchronized

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

CAATS Timeline

Nov. 1989	Treasury Board approves CAATS. Primary contractor awarded fixed-price contract with ceiling of $465.6 million.
1990	Transport Canada rejects first major milestone (contractor's description of proposed new system)
Dec. 1994	Agreement on requirements still not reached. Progress payments stopped ($230 million already paid out)
Sept. 1995	Treasury Board approves substantially amended project and contract
Sept. 1996	CAATS project office and contractor renegotiated contract again, to bring status of project into good standing
Dec. 2000	CAATS delivered to NAV Canada

Auditor General Office, *Systems Under Development: Getting Results*, 1996.
http://www.oag-bvg.gc.ca/internet/English/parl_oag_199611_24_e_5055.html

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Requirements Problems

- Hughes had no local workforce in Vancouver (moved people from their LA base)
- Hughes had little experience with air traffic control
- Expertise of traffic controllers who worked on site became dated
- Scope of project from 1990 - 1995 grew 82%, with over 5300 requirements change requests

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Ineffective Risk Management

A report by the Auditor General of Canada found that project risks were not being managed effectively:

- Government project leaders were part-time, inexperienced
  - lacked knowledge of project management
  - lacked knowledge of system/software development processes
  - failed to track status of project
- Project leaders did not take a central role in decision-making.
  - contractor would propose system specifications
  - Transport Canada would reject then due to nonconformance to its understanding of the requirements

Auditor General Office, *Systems Under Development: Getting Results*, 1996.
http://www.oag-bvg.gc.ca/internet/English/parl_oag_199611_24_e_5055.html

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Ineffective Risk Management

A report by the Auditor General of Canada found that project risks were not being managed effectively (cont.):

- Government entered into long-term, fixed-price contracts before a clear understanding of what will be built was reached by all parties.
- Because contracts were fixed-price, only the contractor's performance with respect to the planned schedule was tracked.
  - Project costs increased dramatically even though contract costs remained fixed
  - Led to compromises in final schedule and final functionality

Auditor General Office, *Systems Under Development: Getting Results*, 1996.
http://www.oag-bvg.gc.ca/internet/English/parl_oag_199611_24_e_5055.html

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Risk Management

- Risk - What could go wrong?
- Risk Analysis
- Risk Mitigation
- CAATS Case Study

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

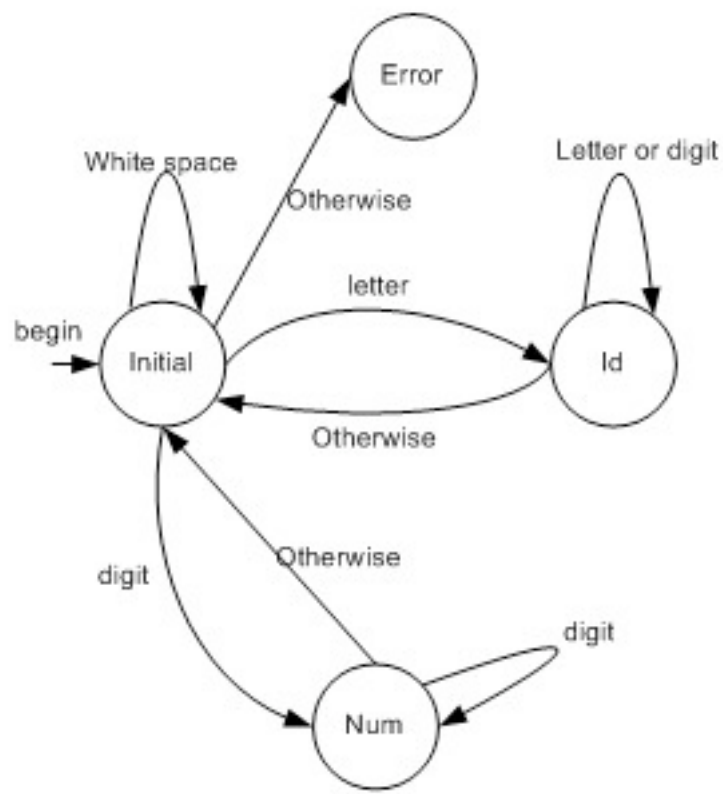
Behaviour Modelling

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Finite State Machines

- You have probably seen an FSM like this to represent a grammar
  - This FSM recognizes a token stream of identifiers and numbers

- "Finite State Automaton" (FSA) is another term for FSM.



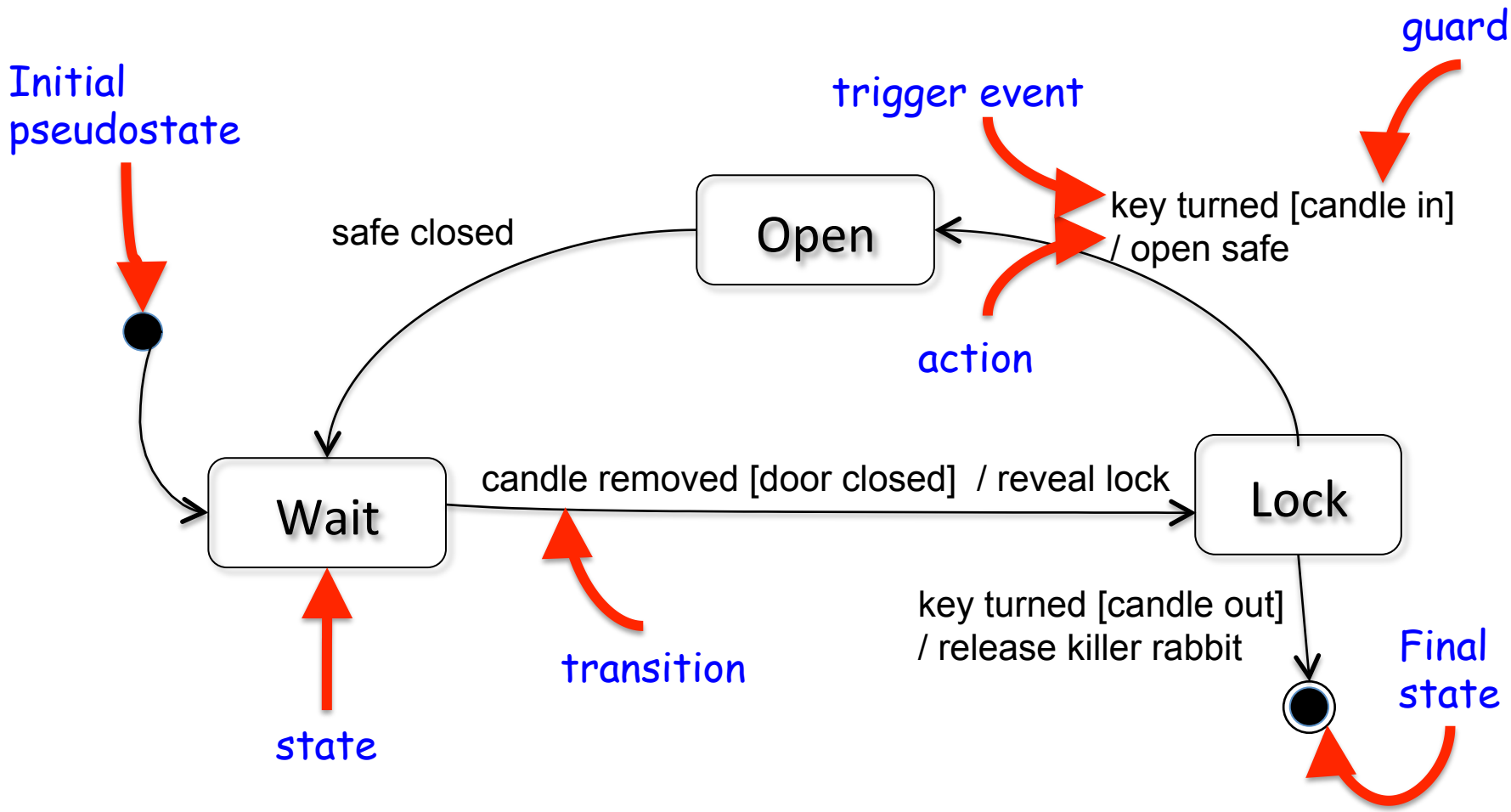
http://www.codeproject.com/KB/recipes/Parser_Expression.aspx

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

State Machines

Fowler, UML Distilled,3ed, p.76

A state machine models behaviour that depends on the history of inputs received so far:

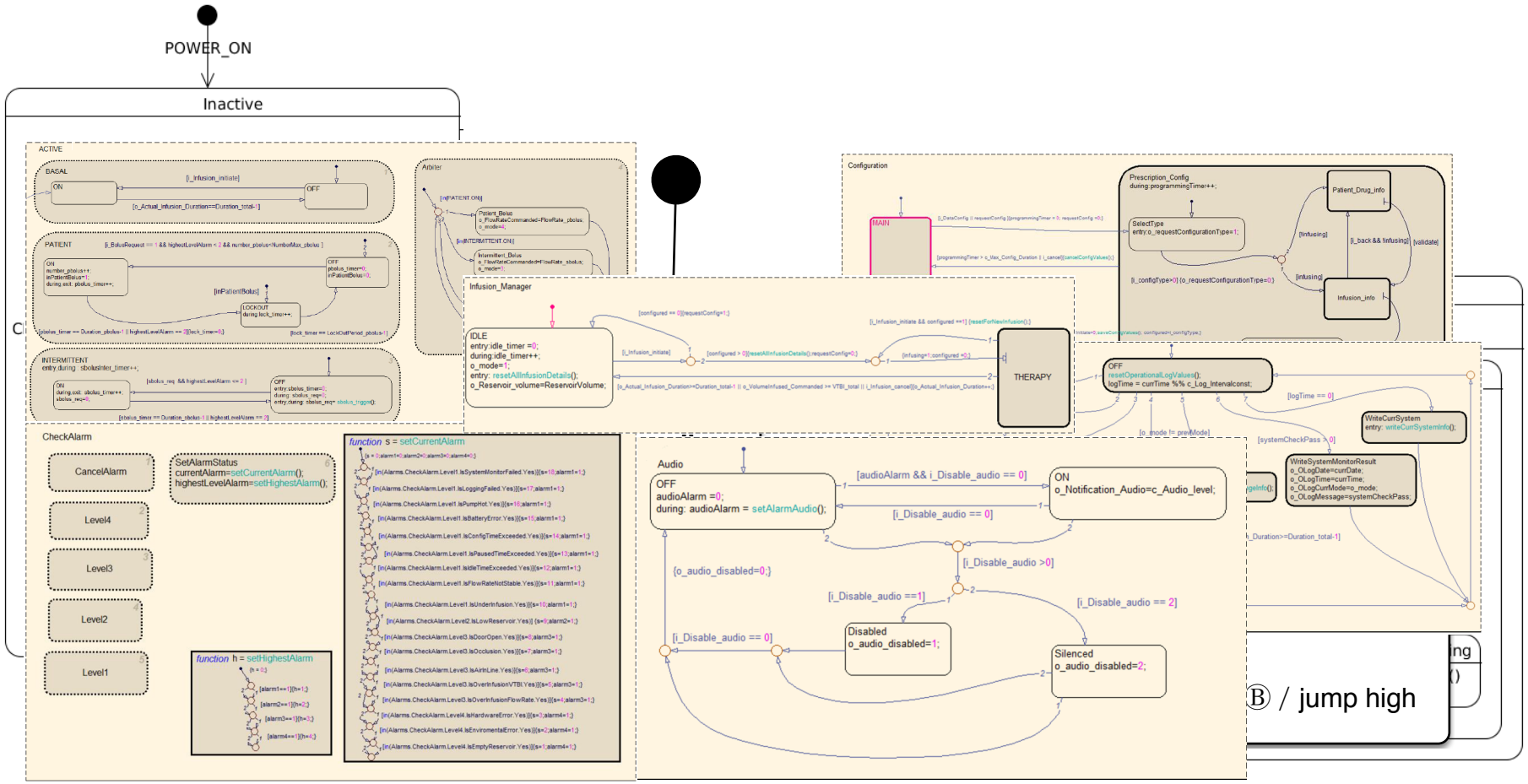


Copyright © Atlee, Berry, Day, Godfrey 1995-2016

State Machines in Practice

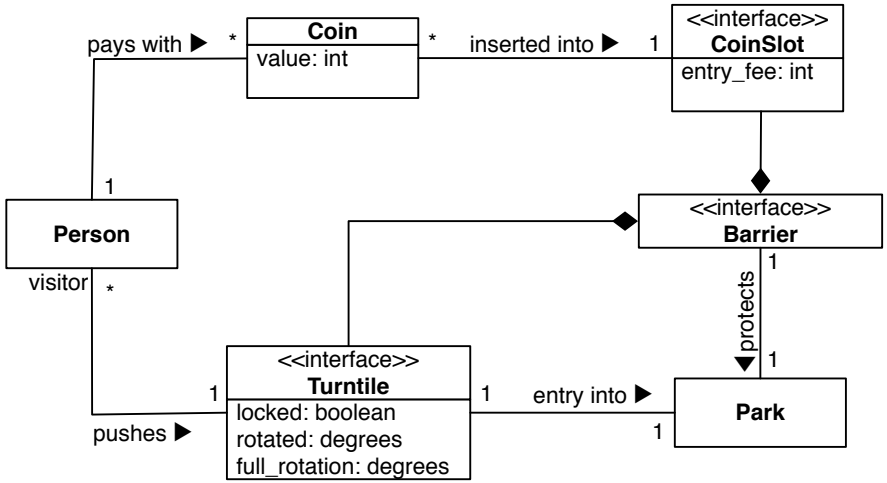
Used to understand complex behaviour

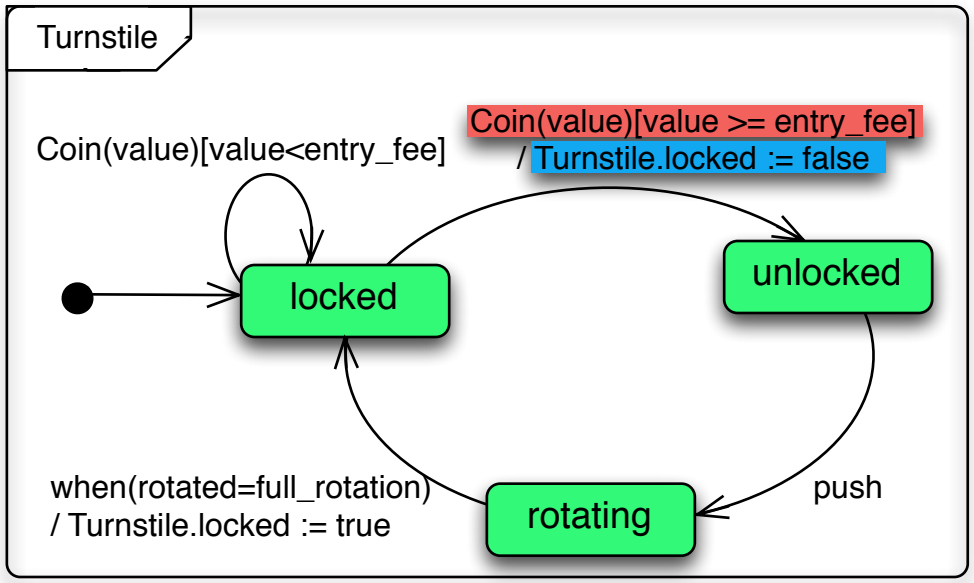
- multiple sequences, cases, conditions



UML State Machines

- States represent equivalence classes of input traces
- Inputs are events and conditions on ≪interface≫ phenomena
- Outputs are actions on ≪interface≫ phenomena
- State machines can have internal variables (not shown)





Copyright © Atlee, Berry, Day, Godfrey 1995-2016

States

- States partition the behaviour of the system
- Each state represents a possible mode of operation, such as
  - equivalence set of history of inputs
  - equivalence set of future behaviours
    - distinct set of input events of interest
    - distinct reactions to input events
  - internal processing of input (delimited by transitions)

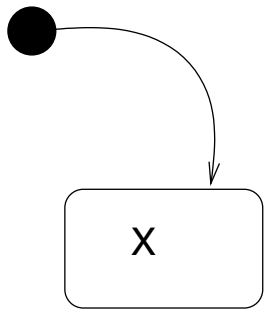
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

States and pseudo-states

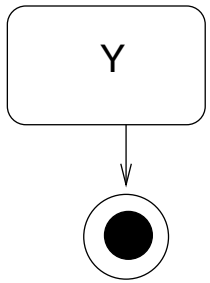
- There always needs to be a designated starting/initial state.

- The designator of an initial state is a *pseudo-state.*
  - A pseudo-state is NOT a real state (no time is spent there)
  - Later, we will see the History pseudo-state

- Often there is a designated final state. This is a real state.



X is the initial State



Bullseye is a final state

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Transitions

Transitions represent observable execution steps

Each of these parts of the transition is optional.

- event(args) — input event / message that triggers the transition
- [condition] — (boolean) guard condition; the transition cannot fire unless the guard condition is *true*
- /action — a simple, fast, non-interruptible action,
  e.g., variable assignment, output event, start complex operation

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Events

An event is a significant or noteworthy change in the environment

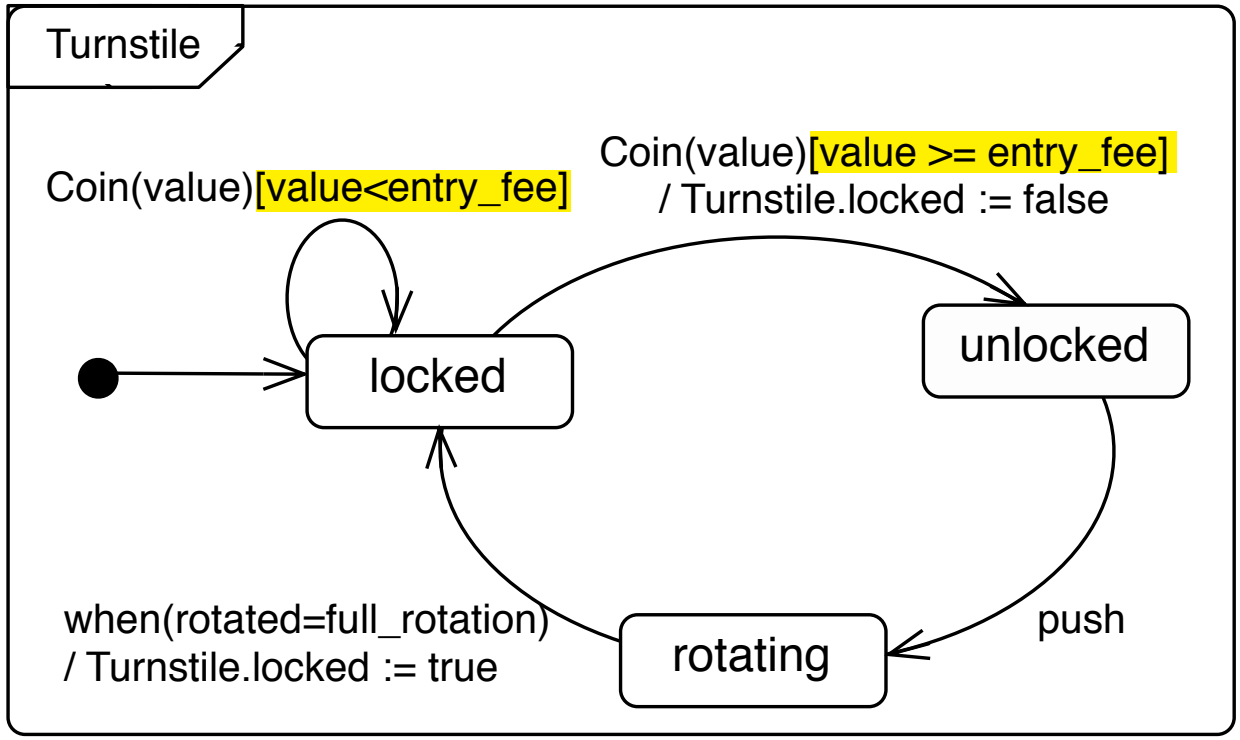
- input message from the environment
  - login request
- some change to ≪interface≫ phenomena
  - e.g., coin inserted, turnstile pushed, elevator button pressed
- passage of time
- multiple events on a transition label are alternative triggers

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Conditions

A condition is a Boolean expression

- over ≪interface≫ phenomena (e.g., [value >= fee])
- over state-machine variables



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Actions

Actions are the system's response to an event

- output message
- change to ≪interface≫ phenomenon
  - e.g., Turnstile.locked := true
  - e.g., AddLoan(m:LibraryMember, p:Publication, today:Date)

An action is non-interruptible (i.e., atomic)

Multiple actions on a transition are separated by ";" and execute sequentially.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Creating a Behaviour Model

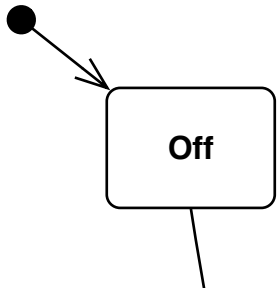
Process:

1. Identify input and output events
   - Changes to interface phenomena in the Domain Model
2. Think of a natural partitioning into states:
   - Activity states - system performs activity or operation
   - Idle states - system waits for input
   - System modes - use different states to distinguish between different reactions to an event
3. Consider the behaviour of the system for each input at each state.

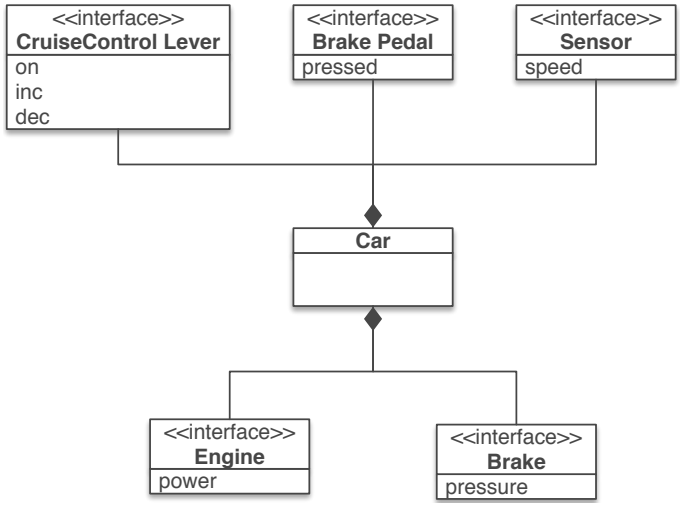
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Cruise Control Example

VARIABLES:
cruise_speed

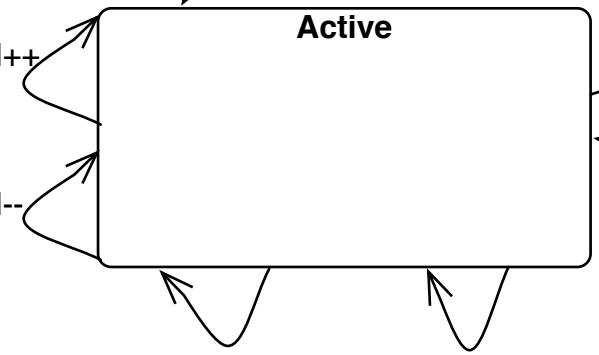


Lever.on
[Sensor.speed ≥ 60 AND NOT BrakePedal.pressed]
/ cruise_speed := Sensor.speed



inc
/cruise_speed++

dec
/cruise_speed--



BrakePedal.pressed

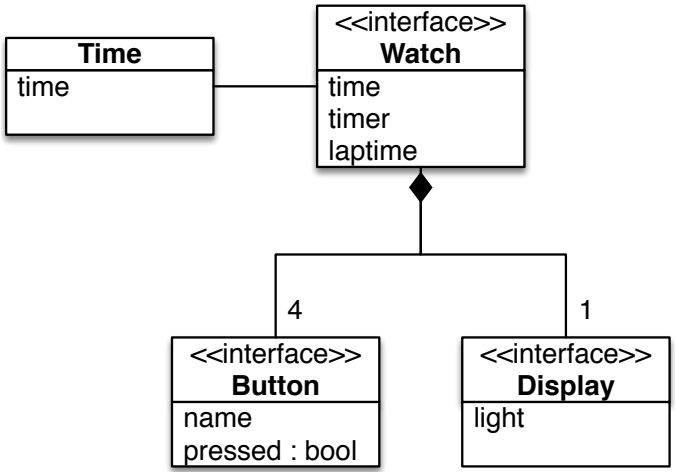
Lever.on
[(| cruise_speed - Sensor.speed | ≤ 10) AND
(NOT BrakePedal.pressed)]

when [cruise_speed < Sensor.speed]
/ increase Engine.power

when [cruise_speed > Sensor.speed]
/ decrease Engine.power

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Stopwatch Example



Input	Meaning of Input Command
on	Turn watch on
off	Turn watch off
mode	Toggle between watch and stopwatch
but2 [watch]	Toggle between 12h and 24h display
but2 [stopwatch]	Pause/ resume timer
but3 [watch]	Turn light on for 3 sec
but3 [stopwatch, timer running, display timer]	Record laptime; display laptime; turn light on for 3 sec
but3 [stopwatch, timer stopped, display timer]	Reset timer; turn light on for 3 sec
but3 [stopwatch, timer running, display laptime]	Display timer; turn light on for 3 sec

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Check for Completeness

For each state X, consider the system's response to each event e:

- There is a transition on e from state X
- Event e cannot physically occur in state X
  - e.g., doorOpened cannot occur when the door is already open
  - no transition on e is needed from X
- Event e is possible but the system should ignore it
  - e.g., multiple "door close" button presses; only first one is significant
  - no transition on e is needed from X
- Event e is possible in state X, but the system should report an error
  - a transition is needed to report error

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

State machine vs. sequence diagrams

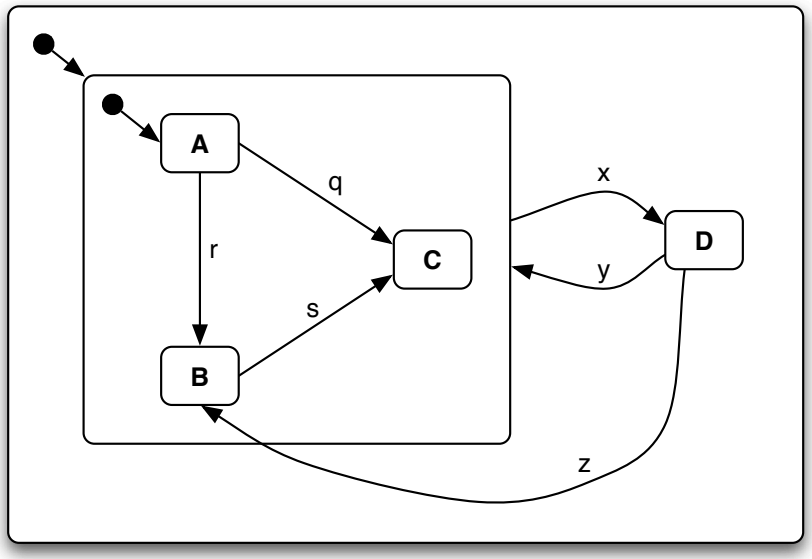
State machine diagrams	Sequence diagrams
specifies behaviour	*Illustrates* behaviour
all allowable scenarios	one allowable scenario, showing end-to-end behaviour (better feel for overall system behaviour)
developer oriented	customer oriented
identifies system states, which represent equivalent input histories	
	can help developer validate state diagrams

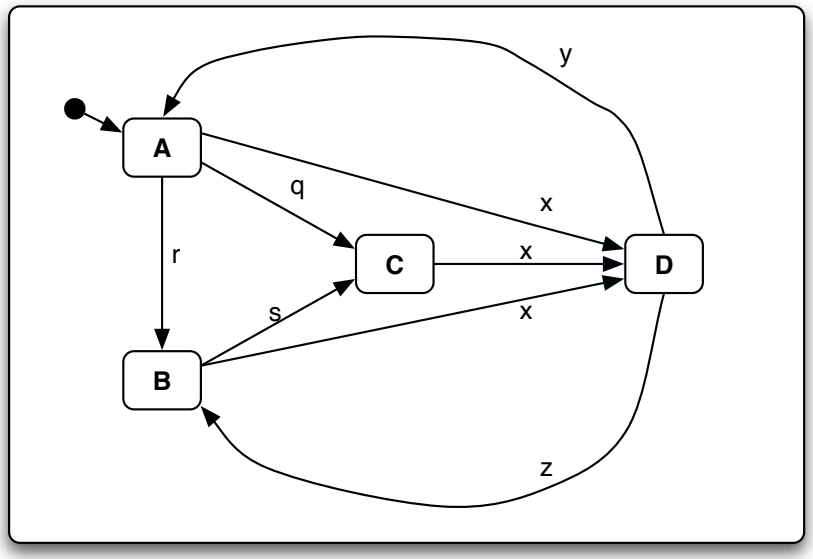
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Hierarchical states

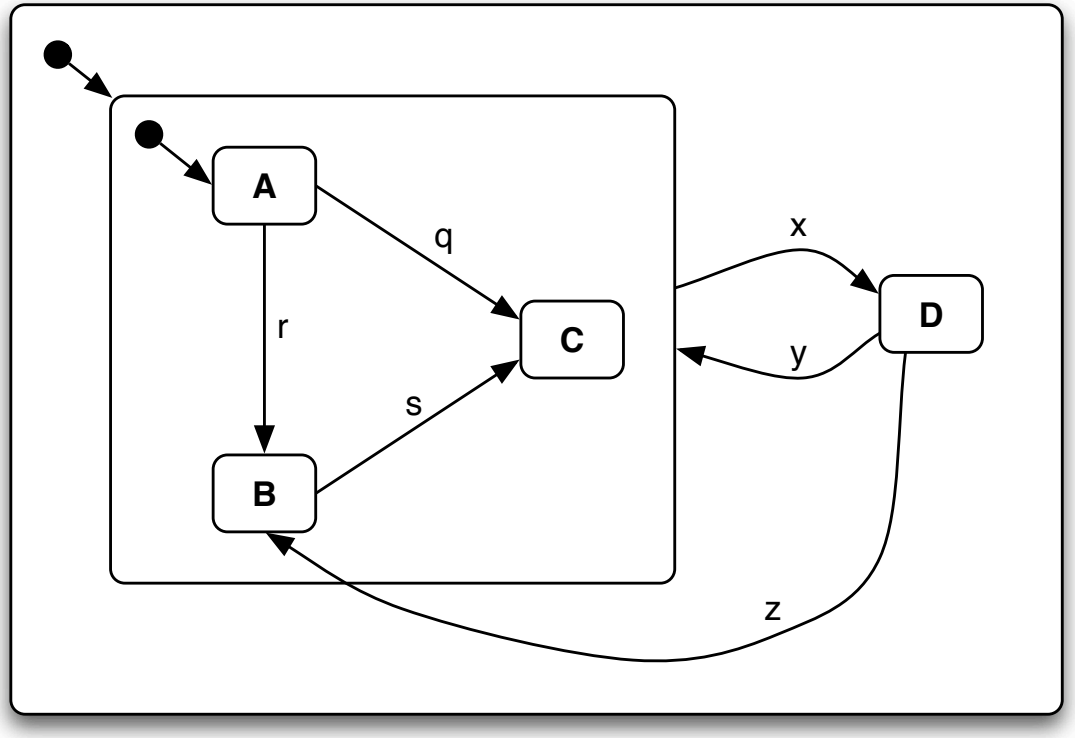
Hierarchy is used to cluster states that have some similar behaviours / exiting transitions.

- One transition leaving a superstate represents a transition from each of the superstate's descendent states.





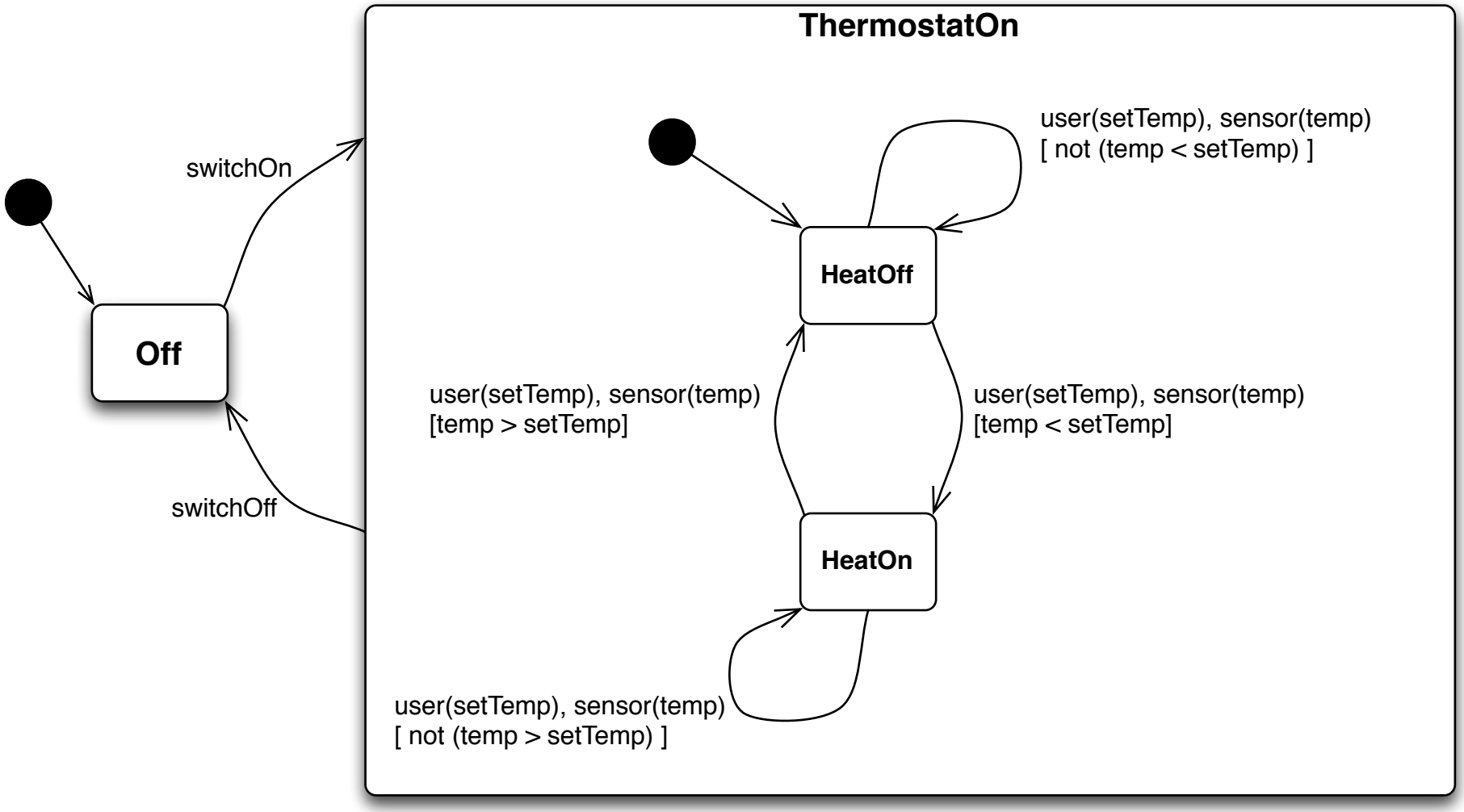
Copyright © Atlee, Berry, Day, Godfrey 1995-2016



- what happens if event z occurs when in state D?
- what happens if event y occurs when in state D?
- can the execution ever leave state C?

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

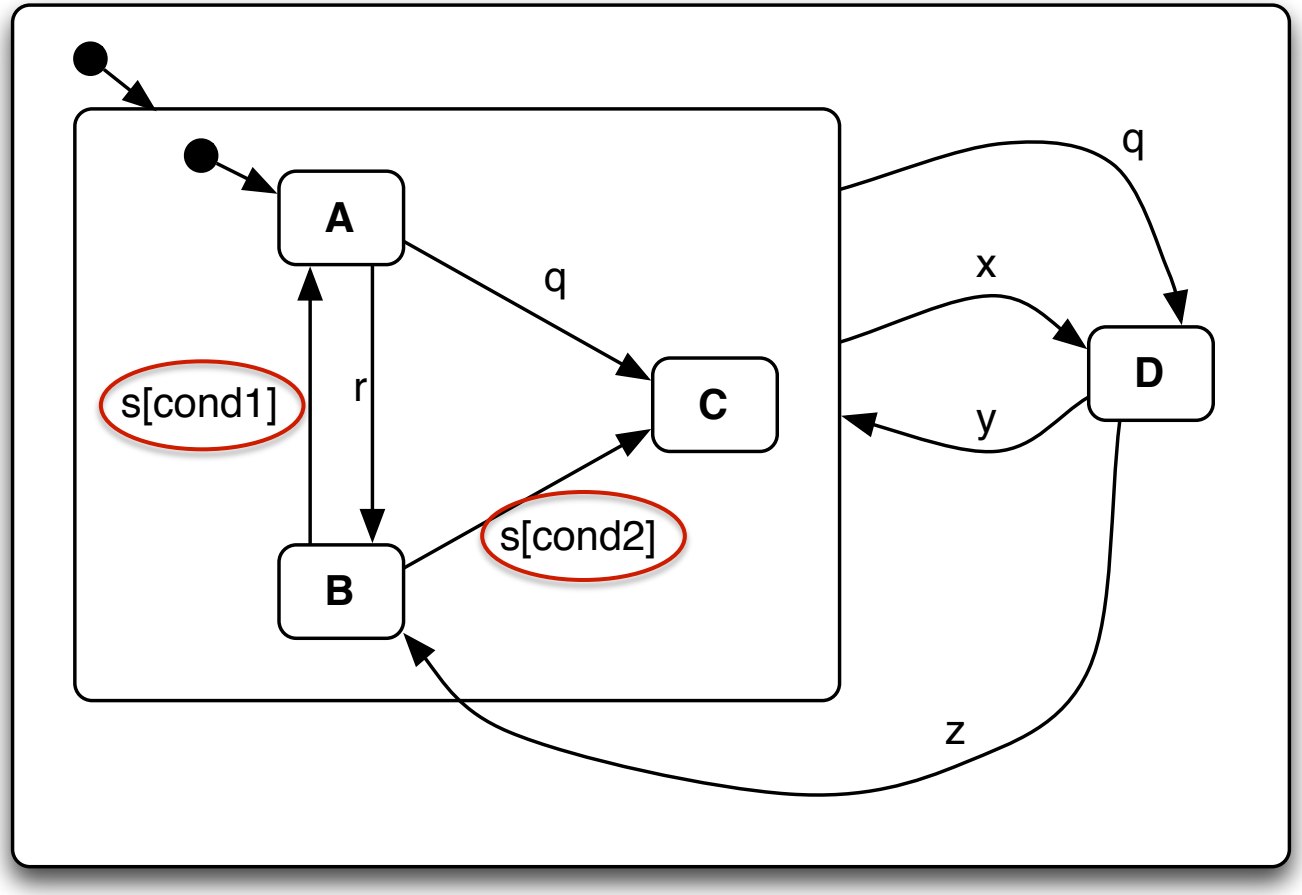
Thermostat example



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Determinism

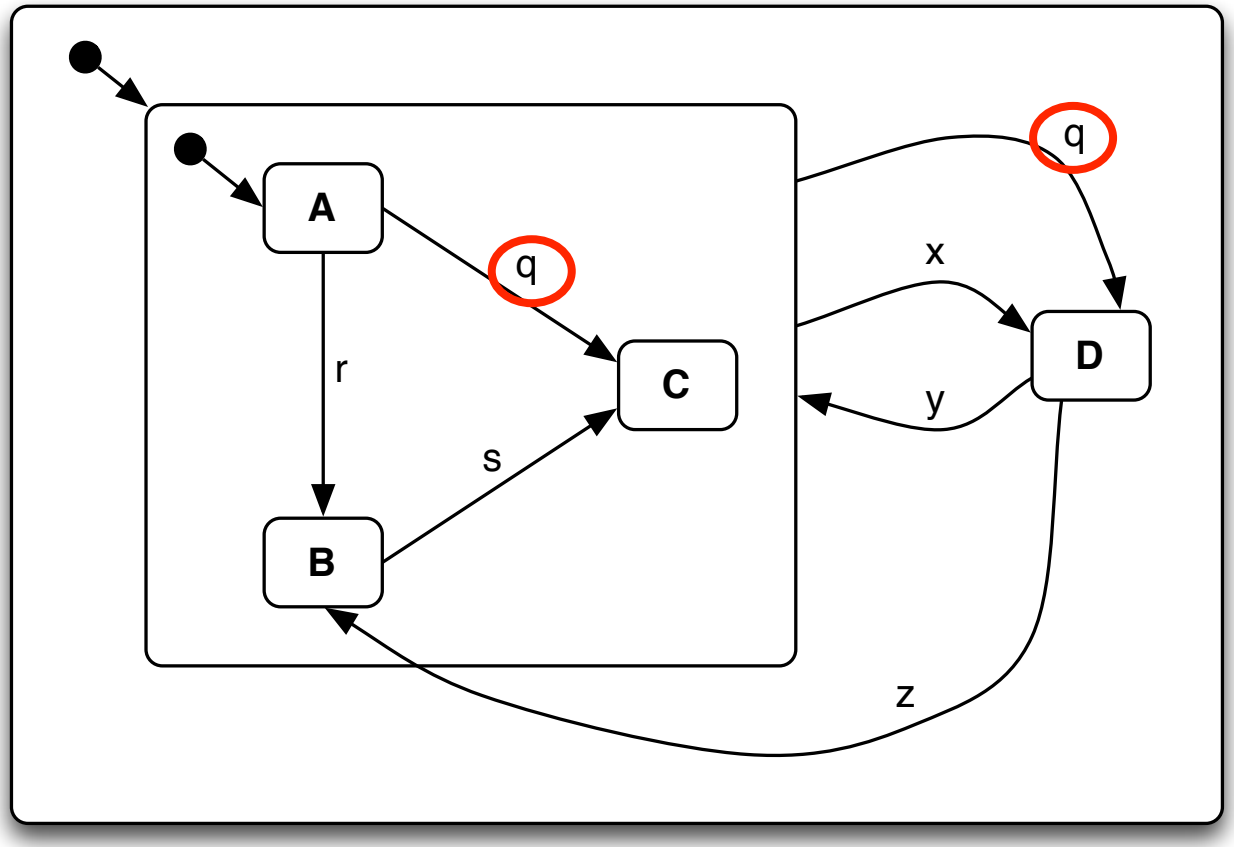
What if the machine is in state B and event s occurs?



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Priority

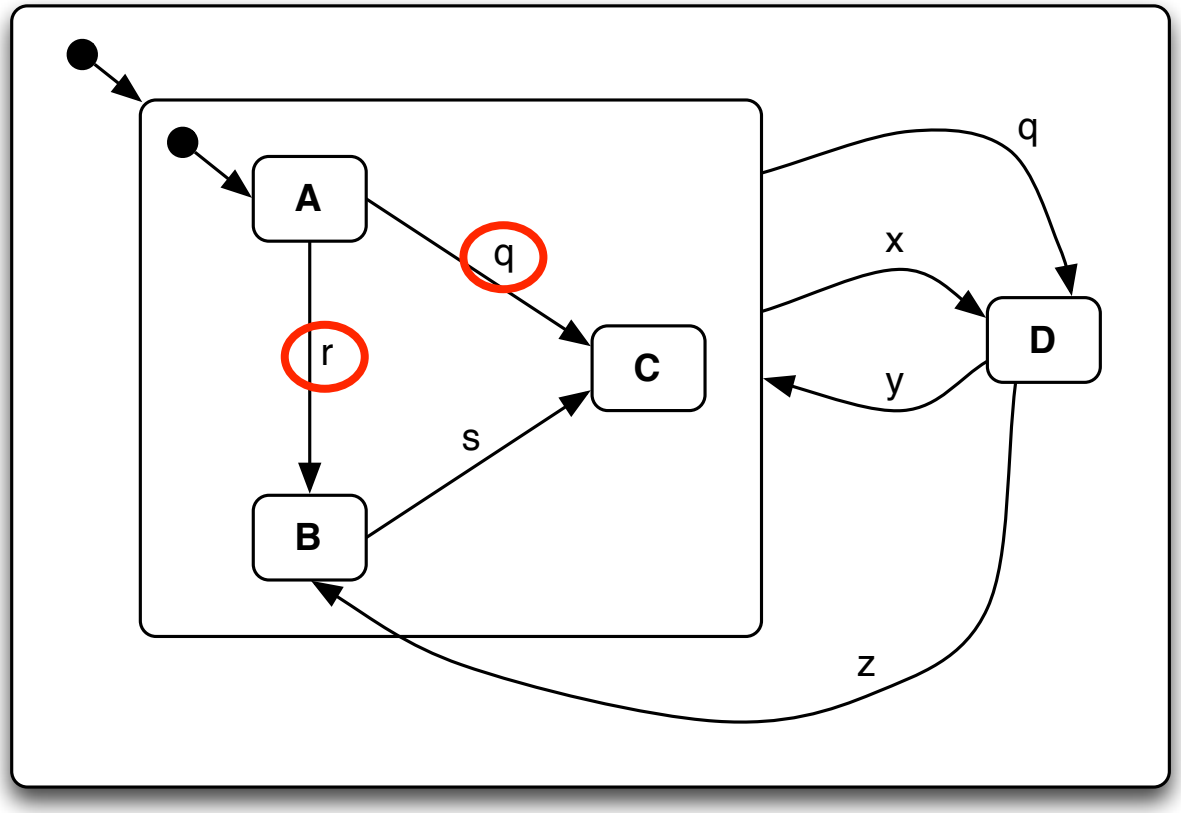
What if the machine is in state A and event q occurs?



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

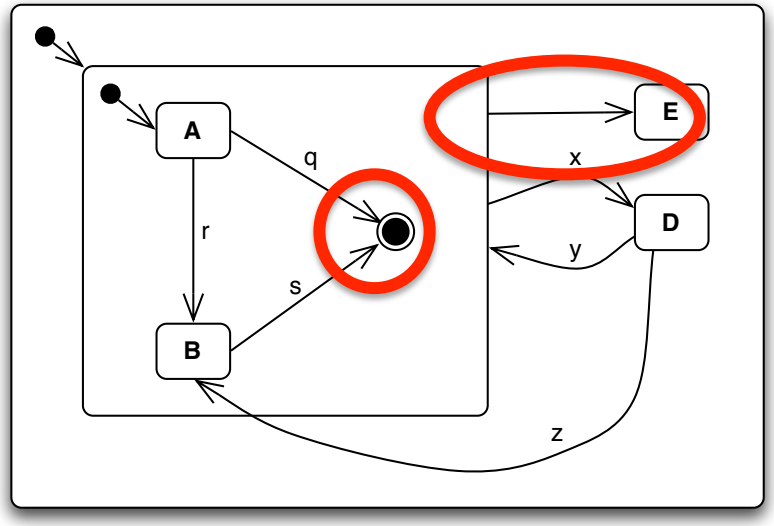
Priority

What if the machine is in state A and events q and r occur simultaneously?



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Final state



A final state represents the end of computation within a superstate.

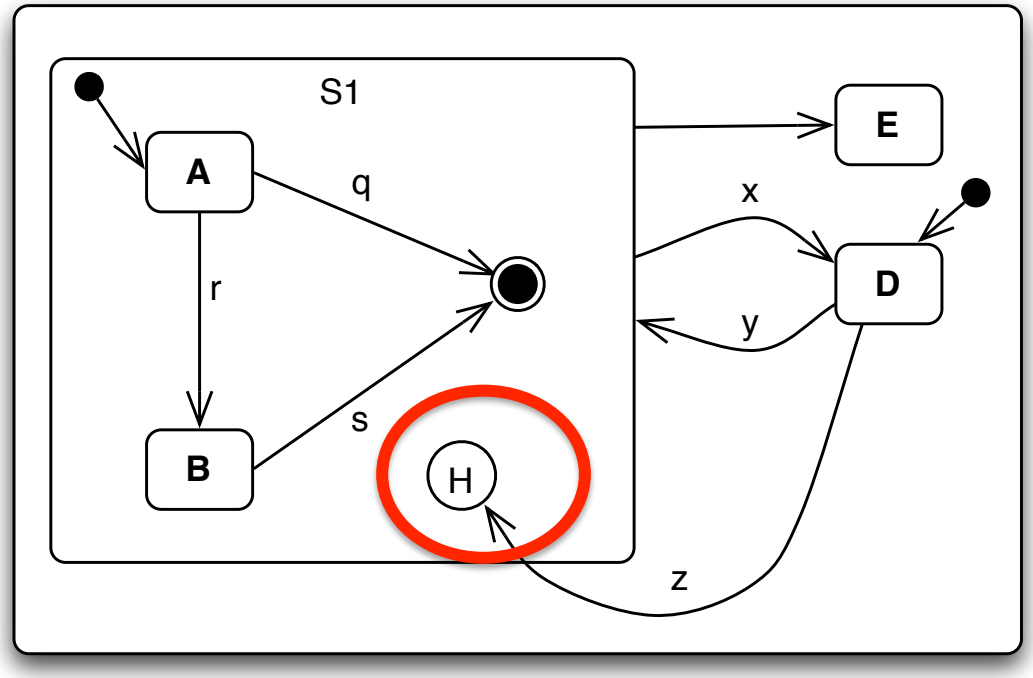
A transition that has no event or condition in its label is enabled when its source state is idle.

- source state is a basic state
- source superstate entered its final state
- source basic state has finished internal activity

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

History

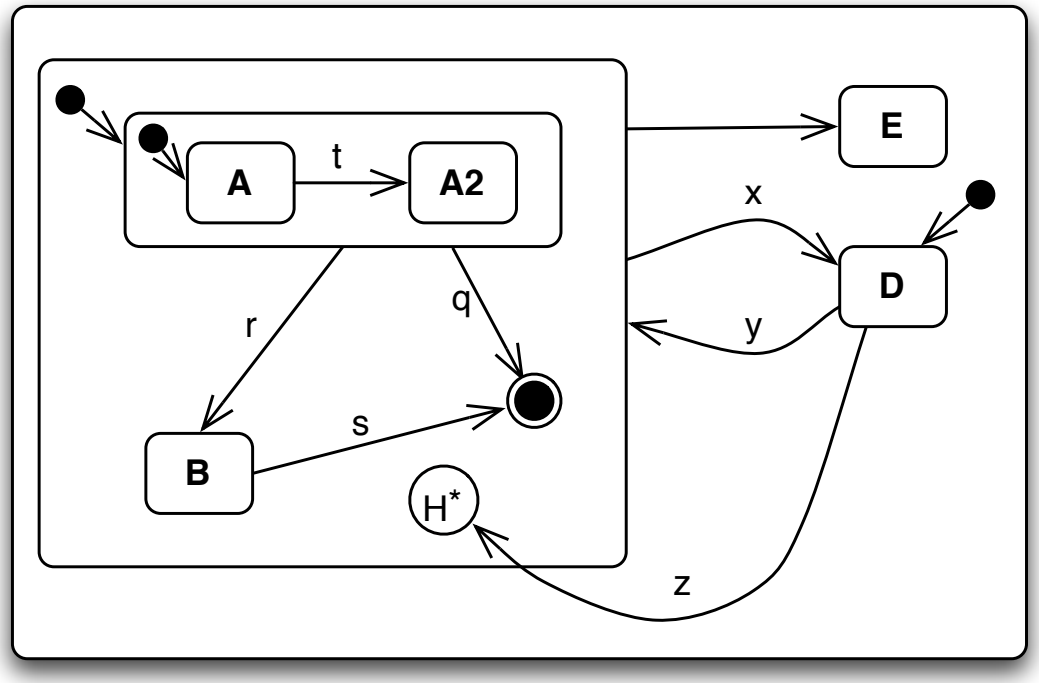
History (H) is a pseudo-state that designates the child state of (H)'s parent state (i.e., child state of S1) that the execution was in when the parent state (S1) was last exited.



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Deep history: H*

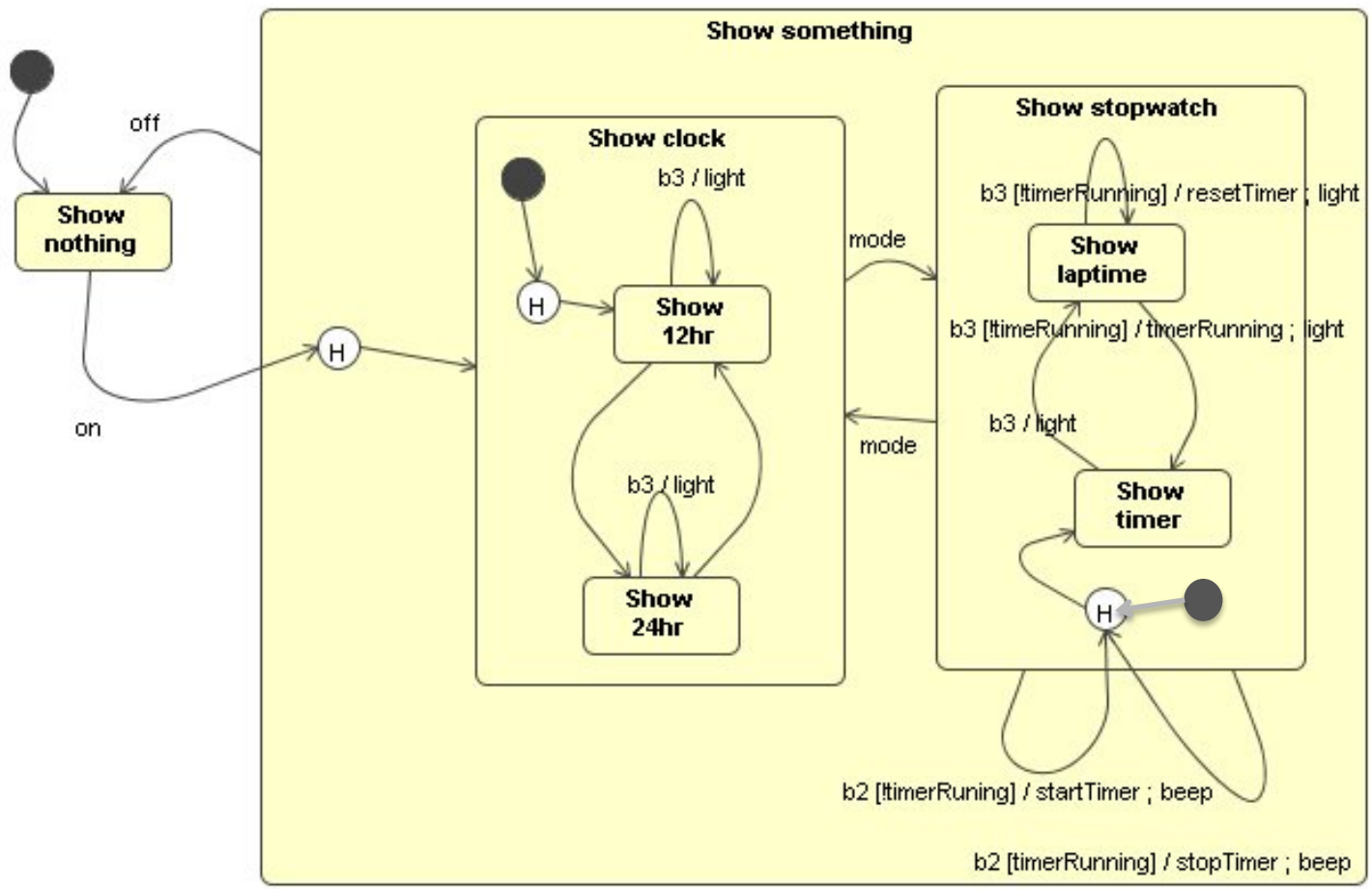
Deep history H* is a pseudo-state that designates the descendant state of H*'s parent state (i.e., descendent state of S1) that the execution was in the when the parent state (S1) was last existed.



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

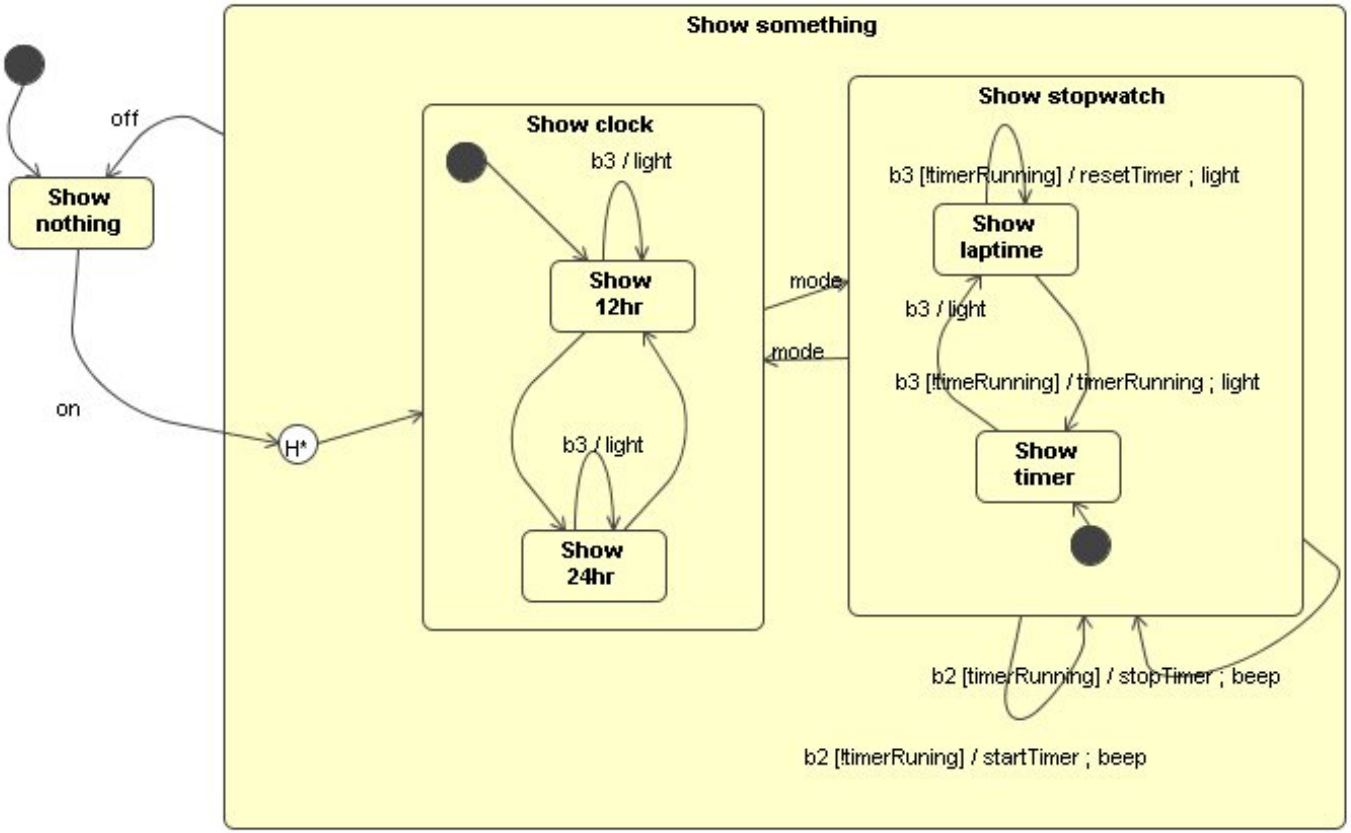
Stopwatch Example

What state is the watch in when it is turned on?



Stopwatch Example

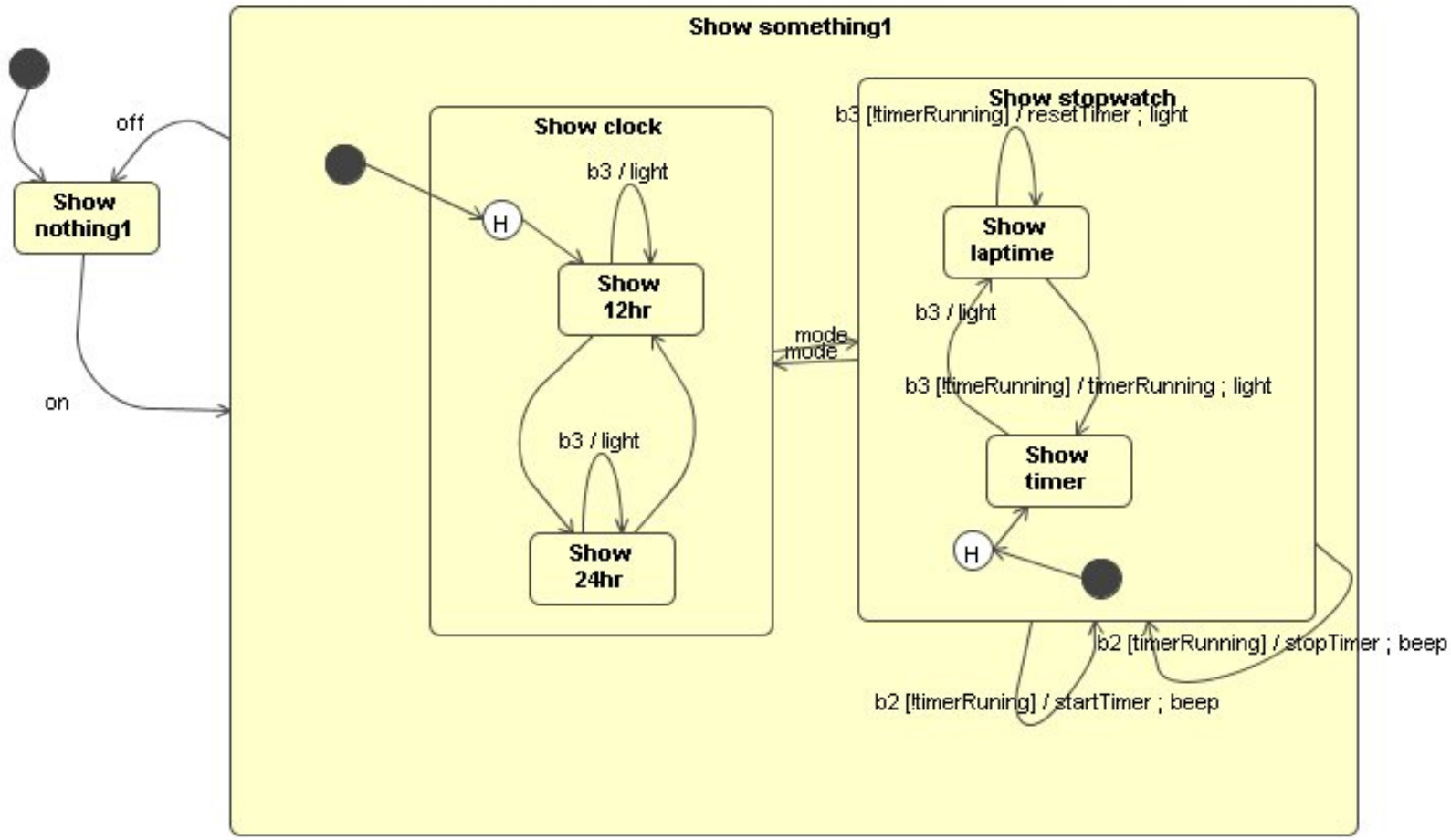
Is this model equivalent to the previous model (i.e., do they represent the same set of execution traces)?



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Stopwatch Example

What state is the watch in when it is turned on?

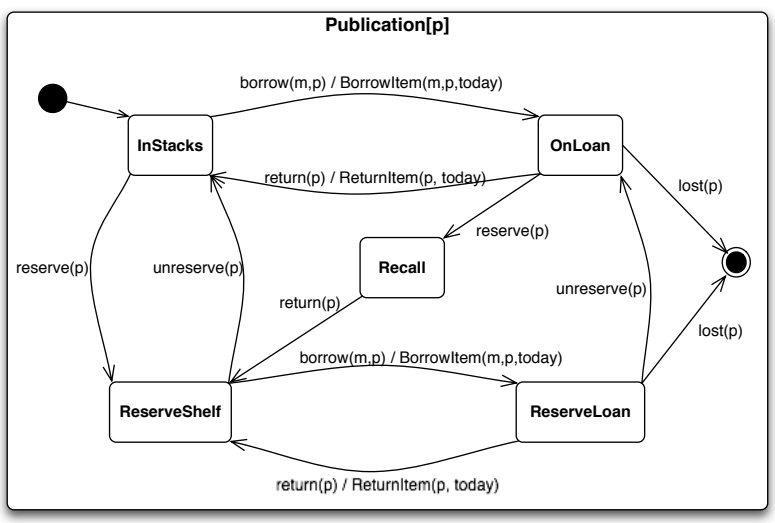


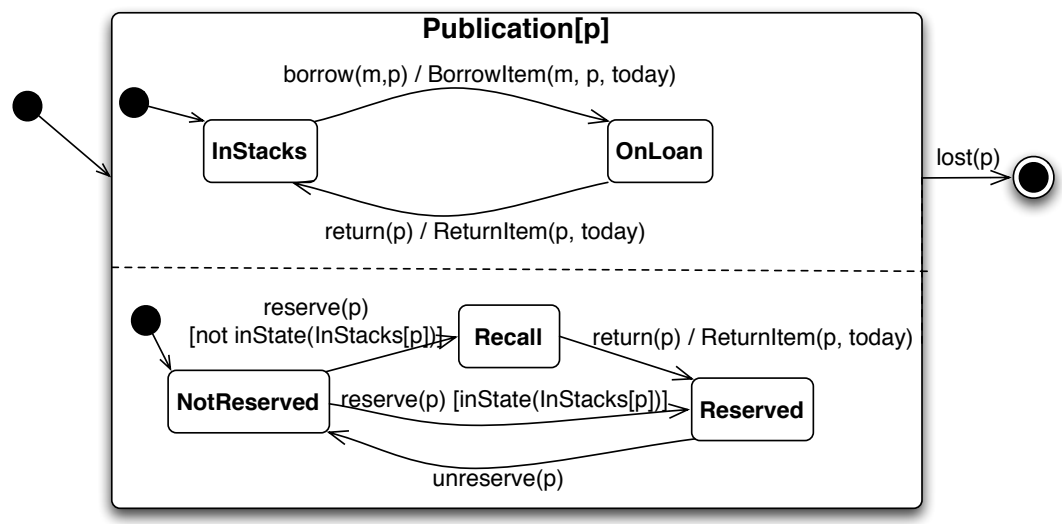
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Concurrent regions

Some systems have orthogonal behaviours that are best modelled as concurrent state machines

- *Regions* within a concurrent state execute in parallel.
- Each has its own thread of control.
- Each can “see” and react to events /conditions in the world

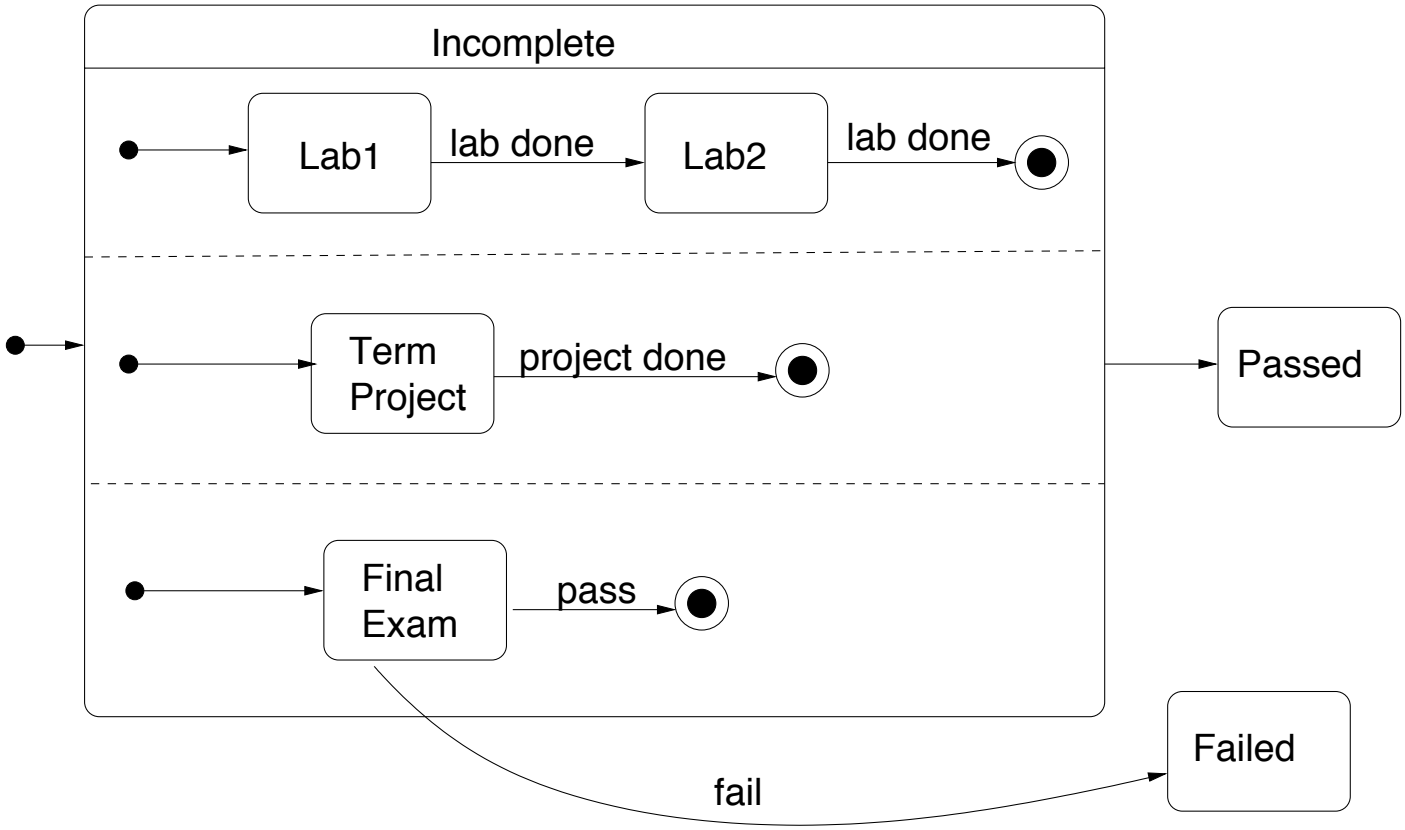




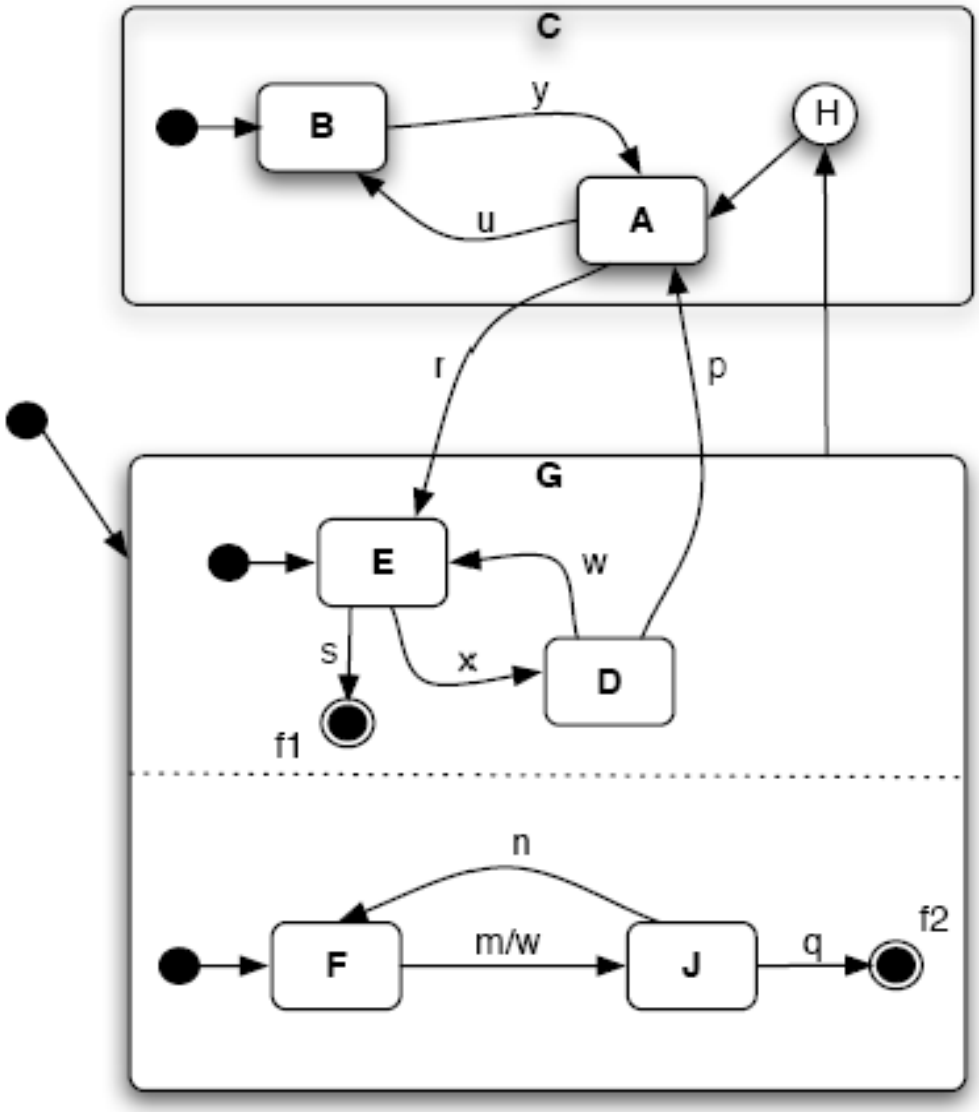
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Concurrency and Final States

Execution of a concurrent state is considered finished when all its regions are in their final states.



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

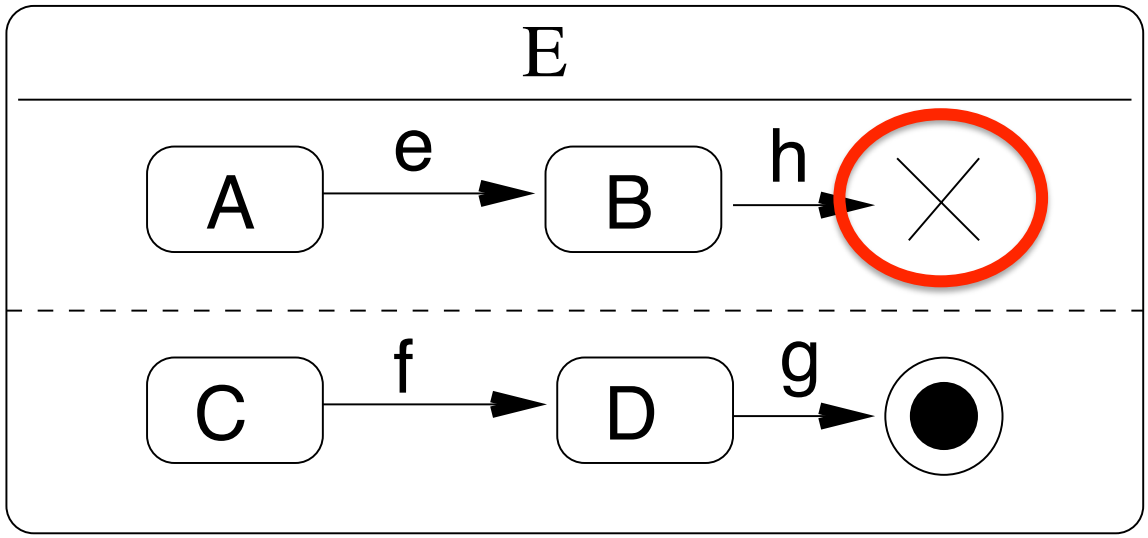


Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Termination

To represent the end of system execution, use the termination state.

- It's like a poison pill for the machine
- Not the same as a final state in a region, which waits patiently for the other regions to finish



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Coordination of Regions

Ways to coordinate regions with each other

- Regions react to the same input event
- Regions react to the value of the same ≪interface≫ phenomenon
- One region generates an event that other regions react to
- A region's transition has guard [inState(x)] that evaluates to *true* iff another region is in state *x,*
  - [*NOT part of UML, just a useful convention!*]

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Time event

A time event is the occurrence of a specific date/ time or the passage of time.

- Absolute time:
  - at (12:12 pm, 12 Dec 2012)
- Relative time:
  - after (10 seconds since exit from state A)
  - after (10 seconds since x)
  - after (20 minutes) // since the transition's source state was entered

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Change events

A change event is the event of a condition becoming true.

- The event “occurs” when the condition changes value from *false* to *true*.
  - when (temperature > 100 degrees)
  - when (on)
- The event does not reoccur unless the value of the condition becomes *false* and then returns to *true*.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

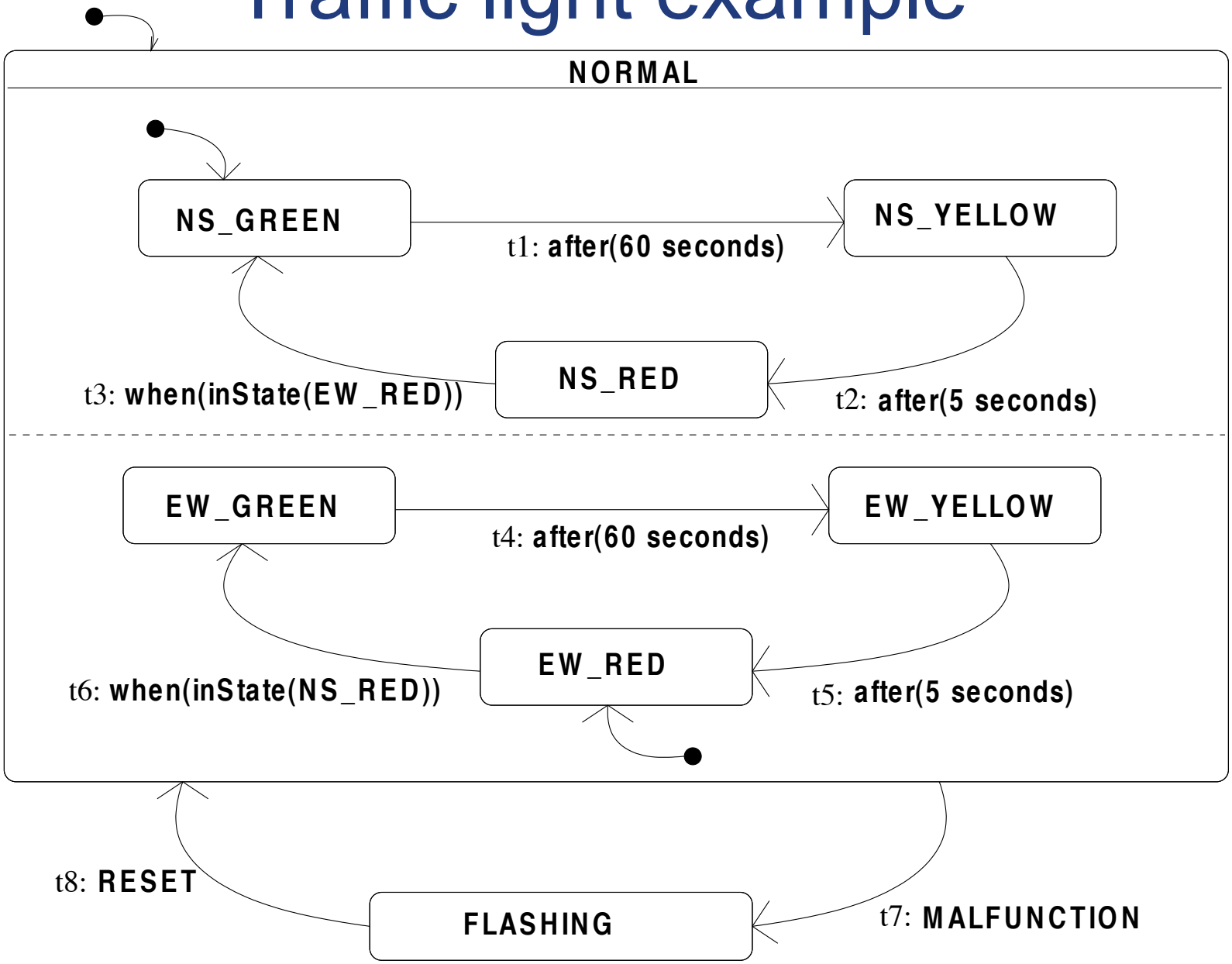
when(X) vs. [X]

- A change event is what you want if you're sitting in a state waiting for a condition to become true

- A transition with a guard but no triggering event
  - called a "completion transition" in UML
  - semantics:
    - guard is checked once (when the source state becomes idle)
    - transition fires if the guard is true
  - guard is not checked again
  - not likely what you want to say

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Traffic light example



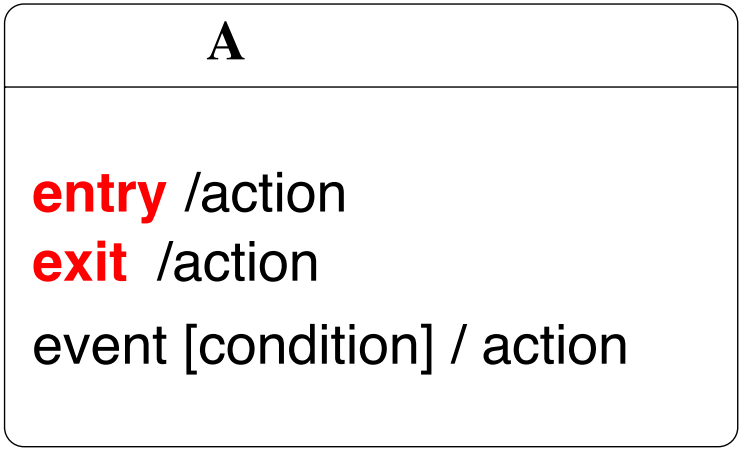
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

State Actions

Recall: An action is uninterruptible.

States can also be annotated with entry or exit actions, and with internal actions.

- Entry actions: – actions that occur every time the state is entered by an explicit transition.
- Exit actions: – actions that occur every time the state is exited by an explicit transition.
- Internal actions: on events

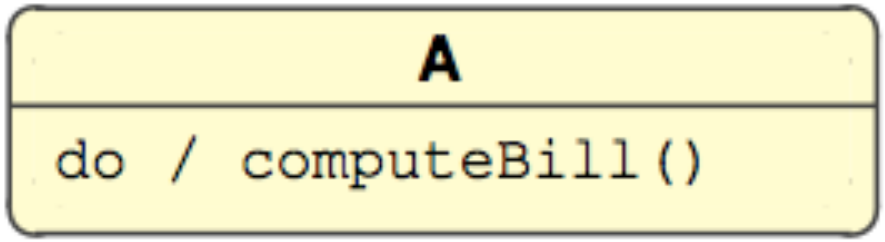


Copyright © Atlee, Berry, Day, Godfrey 1995-2016

State Activities

An activity is computation of the system that takes time, and can be interrupted.

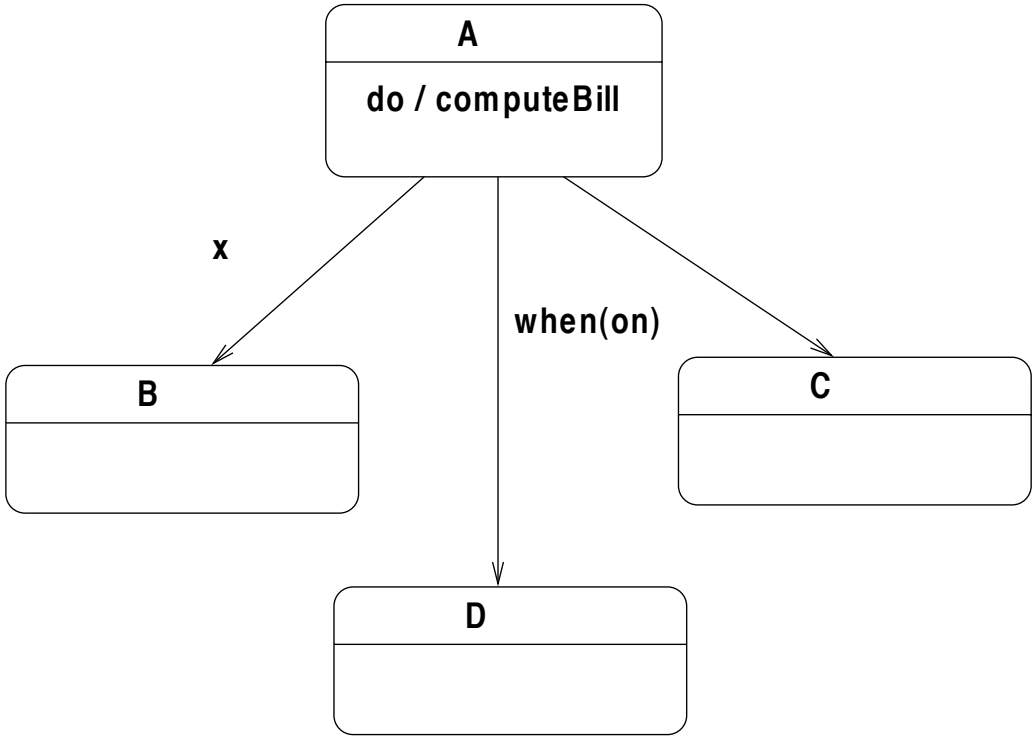
- c.f., an action, which is uninterruptible
- An activity may be associated with a state.
- States with activities are called *activity states*.



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

State Activities

Because activities take time, they can be interrupted by transitions with triggers and/or conditions



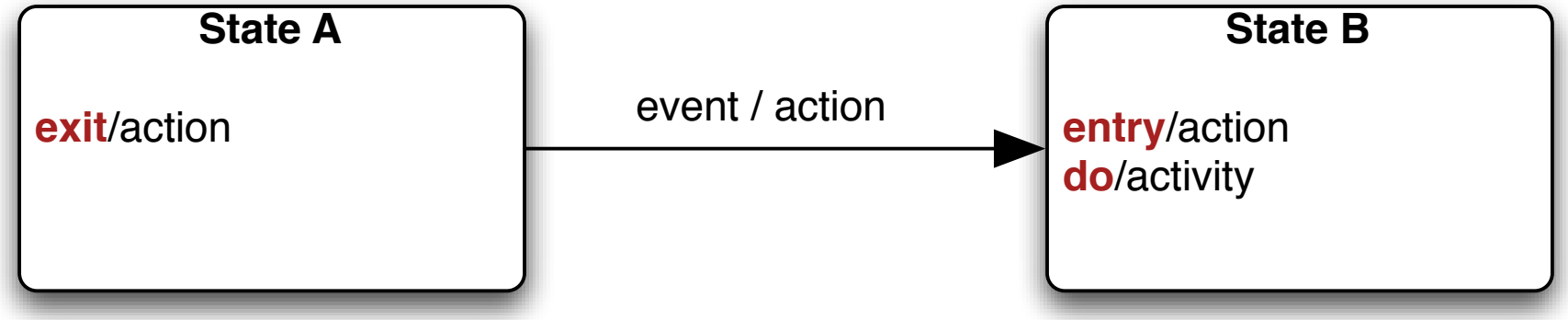
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Actions vs. Activities

- UML 2.0 has dropped actions ☹
  - This is too bad because the synchrony hypothesis is a very useful, valid, and reasonable simplifying assumption for requirements modelling.
  - Therefore, we will use actions in our state machine models anyway. So there.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Execution Order



In an explicit transition (including self-looping transitions!), the order of effects is:

1. exit actions of source state, then
2. transition actions (in listed order), then
3. entry actions of destination state, then
4. state activities.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Creating a Behaviour Model

1. Identify input and output events
2. Think of a natural partitioning into states
   - *Activity states* – system performs activity or operation
   - *Idle states* – system waits for input
   - *System modes* – use different states to distinguish between different reactions to an event
3. Consider the behaviour of the system for each input at each state.
4. Revise (using hierarchy, concurrency, state events)
   - Use concurrency to separate orthogonal behaviour
   - Use hierarchy, and entry/exit actions, to abbreviate common behaviour

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

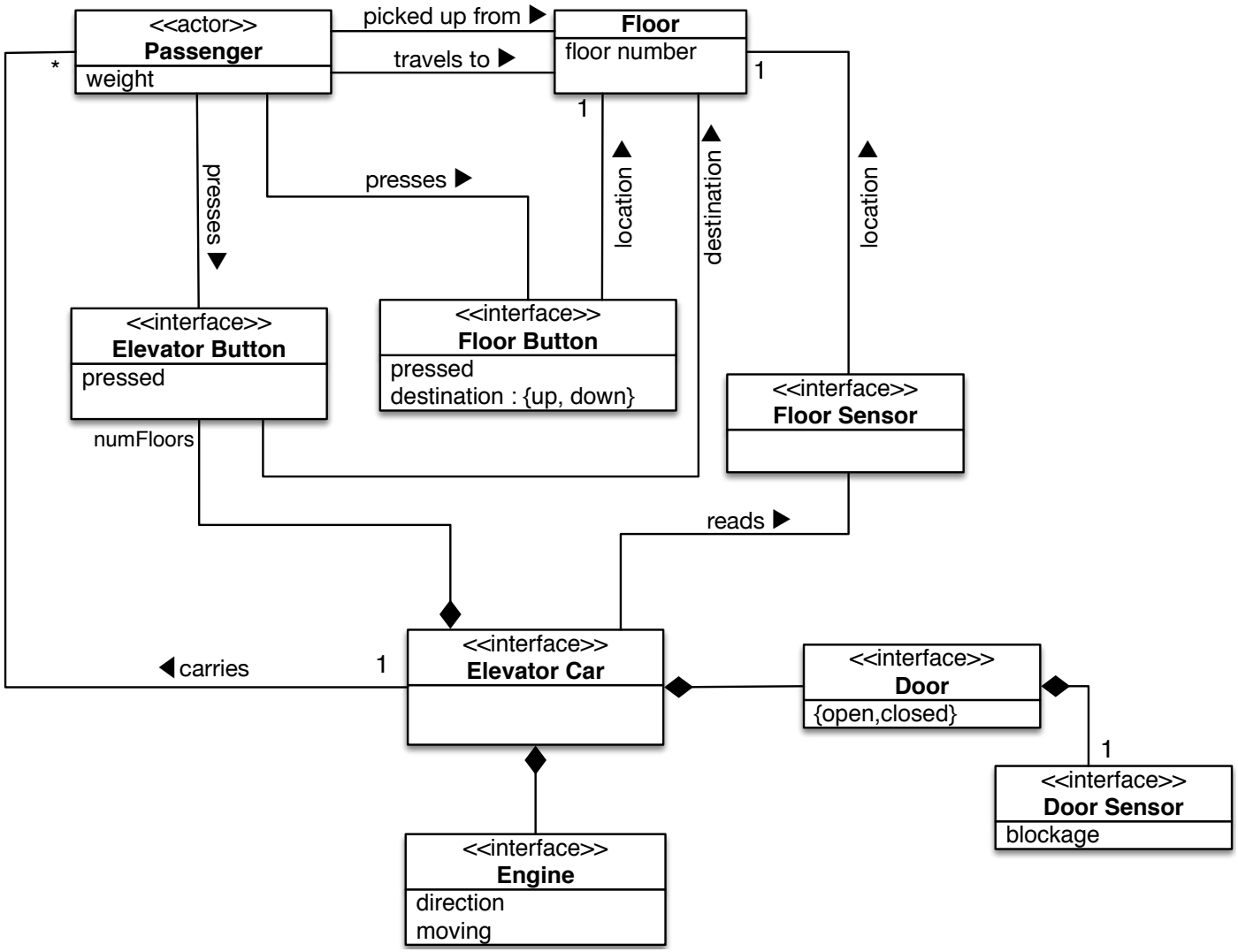
Example: Elevator

- An elevator passenger who wants to travel from one floor to another higher (lower) floor presses the Up (Down button at his current floor.
- The light beside the button must then be lit, if it was not lit before.
- The elevator must arrive reasonably soon, travelling in an upwards direction.
- The direction of travel is indicated by an arrow illuminated when the elevator arrives.
- The doors must open, and stay open long enough for the passenger to enter the elevator.
- The doors must never be open except when the elevator is stationary at a floor.

Michael Jackson, *Software Requirements and Specifications, Addison-Wesley, 1995.*

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Elevator



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Good style

The best state machine model is usually the one that is the clearest.

- Use states to model modes of operation, and use variables to model other information that affect flow of control
- Fewer transitions are better
  - i.e., Use hierarchy to reduce the number of transitions
- Use history + deep history
- Use concurrency to recognize orthogonal aspects of the problem

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Common problems

- Over-specification:
  - Don't build an executable model
    - Model the possible flows of control, not the possible computations
    - e.g., don't keep track of the actual time in stopwatch model

- Under-specification:
  - Do specify a response to every event that is relevant at a state.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Validating Behaviour Models

- Avoid inconsistency: multiple transitions that leave the same state under the same event/conditions.

- Ensure completeness: specify a reaction for every possible input at a state.
  - If there are transitions triggered by an event conditioned on some guard, what happens if the guard is false?

- Walkthrough: compare the behaviour of your state diagrams with the use-case scenarios.
  - All paths through the scenarios should be paths in the state machines.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

State Machine Models

- applied to a world state (of the domain model)
- hierarchy
- concurrency
- state actions / activities

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463

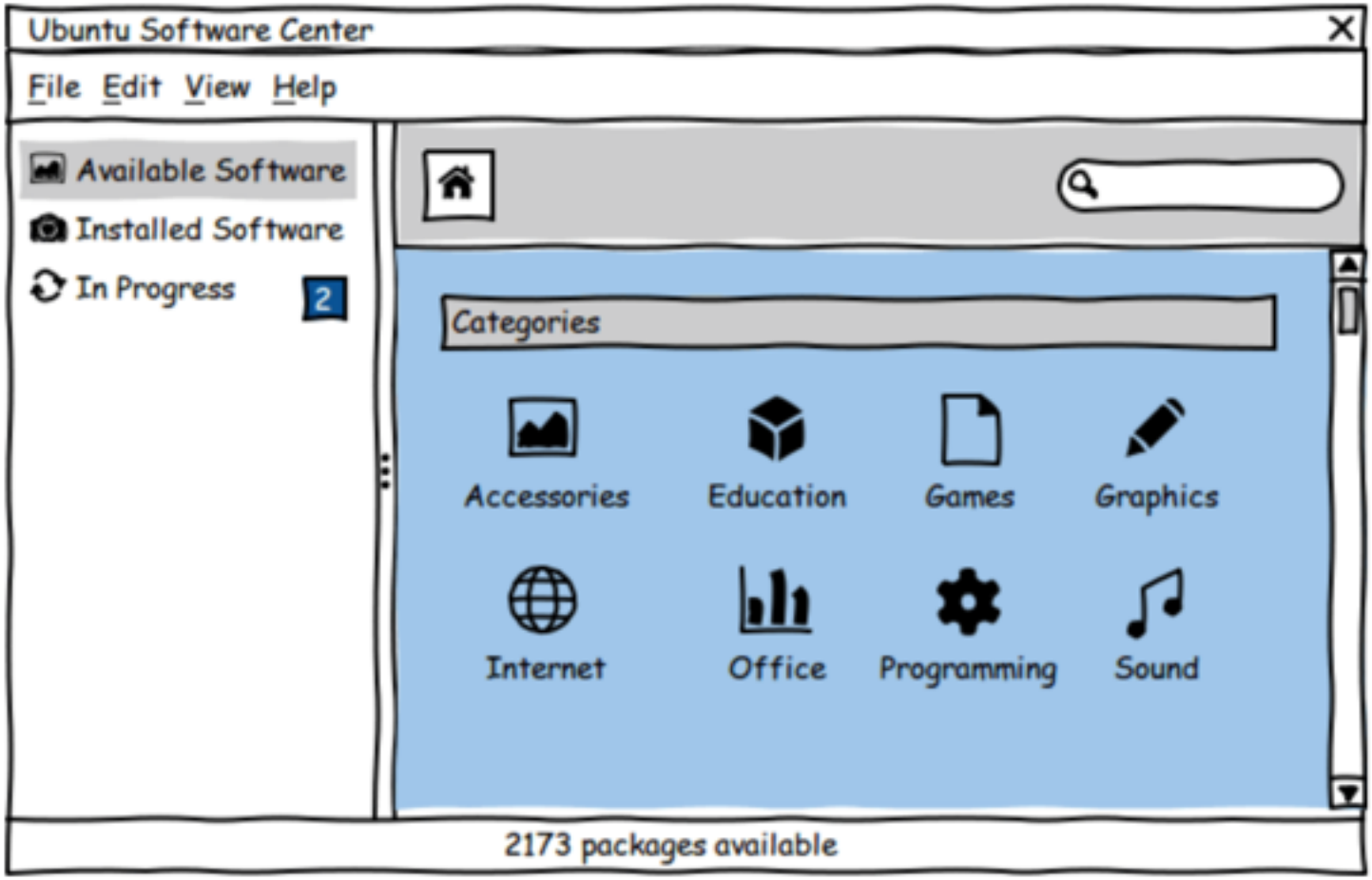
Software Requirements Specification & Analysis

Mockups

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Mockups

Sketch the essence of a solution, and use to bait stakeholders into providing new requirements details



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

User Experience (UX)

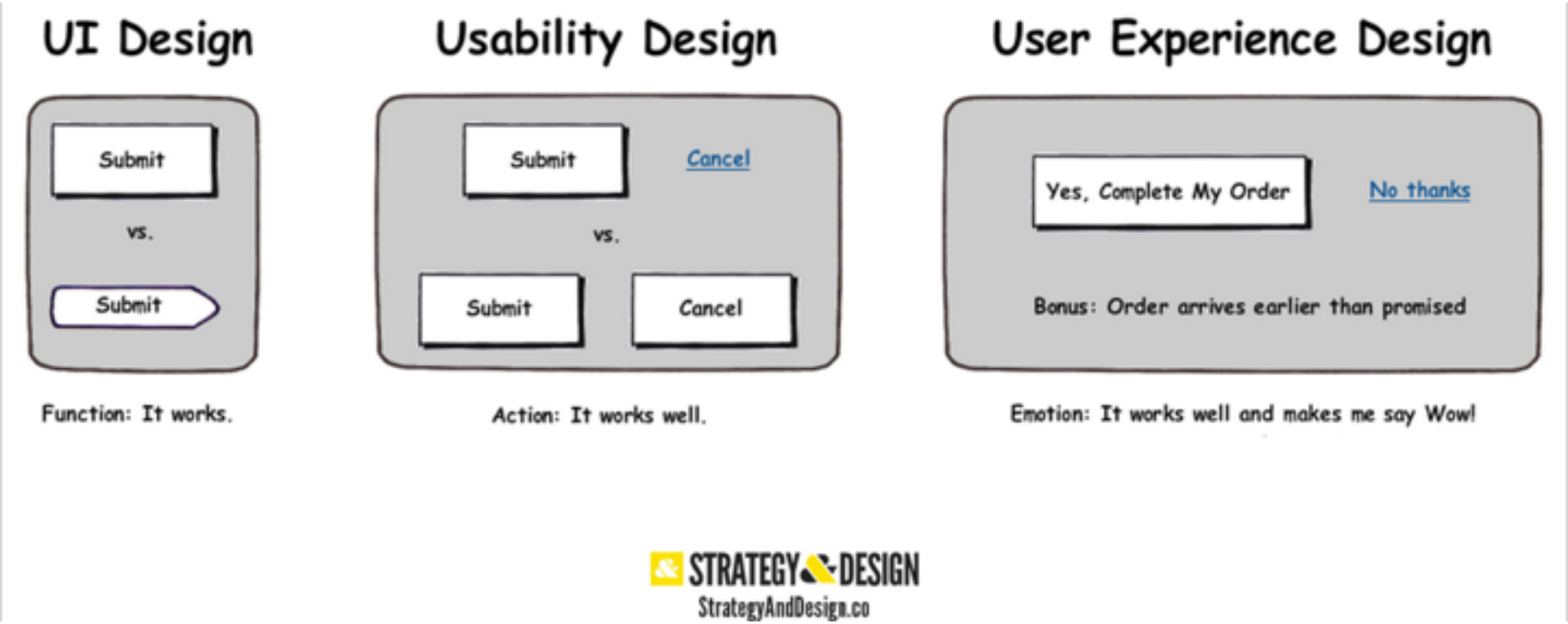
User experience (UX) encompasses all aspects of a user's interaction with a company, its services, and its products.

- Meet the exact needs of the customer
- Be simple, elegant, intuitive – a joy to own and use
- Be a seamless merging of services

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

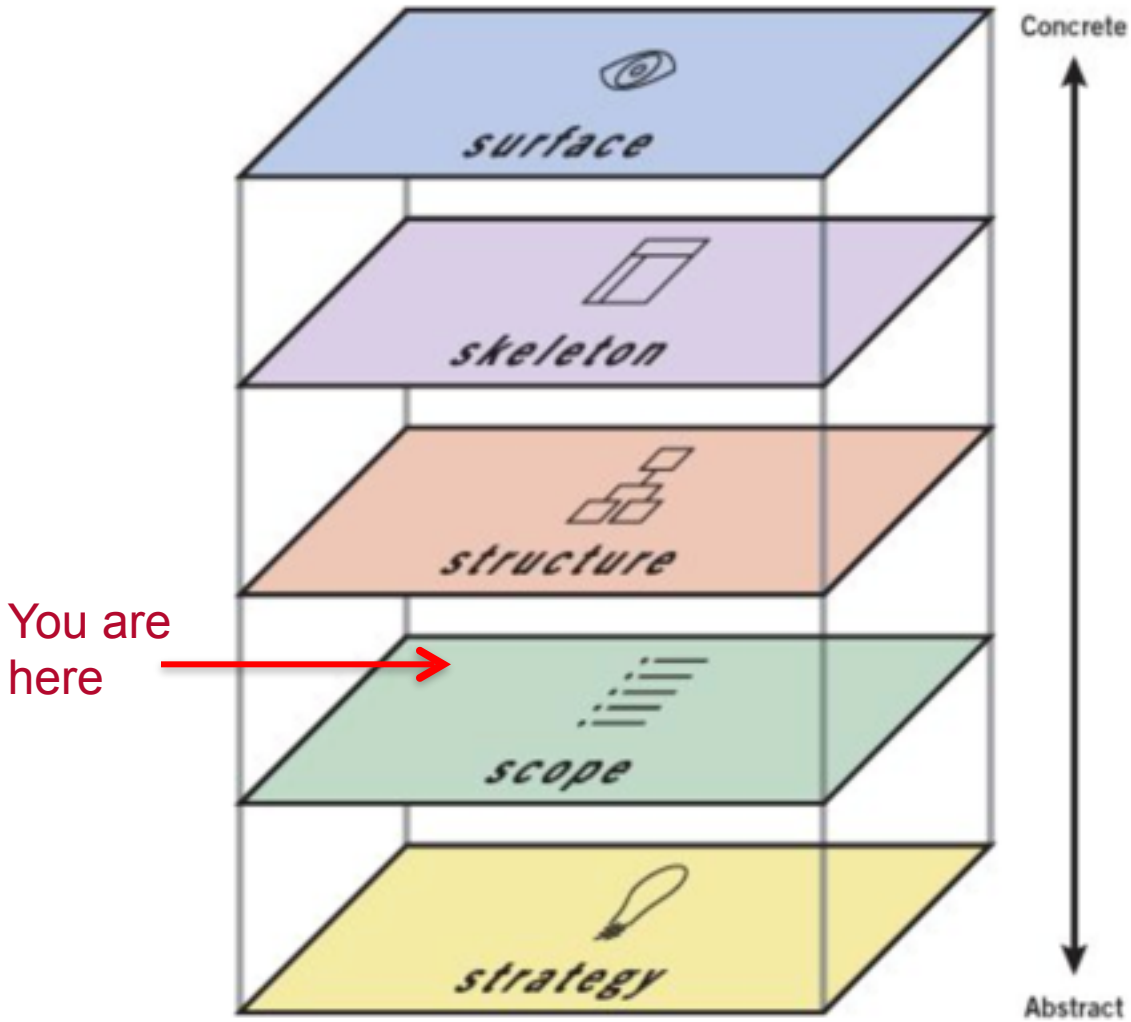
User Experience (UX)

Not to be confused with usability, which is a quality attribute that concerns the product's ease of use, ease to learn, how efficient and productive users are, etc.



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

User Experience



Surface
Graphics
Images

Skeleton
Prototypes
Wireframes

Structure
Sitemaps
Interactions

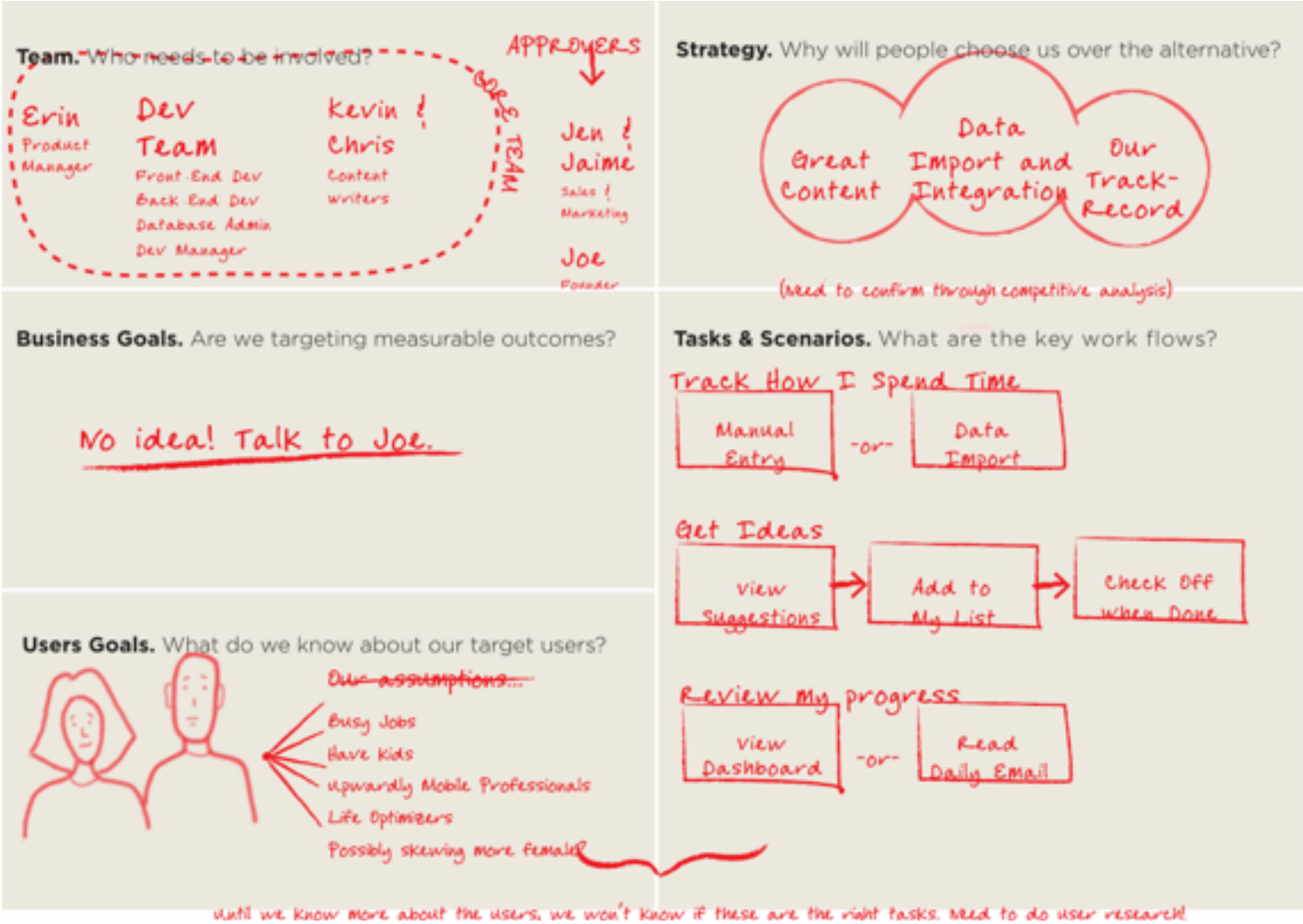
Scope
Requirements
Specifications

Strategy
User needs
Business goals

Jesse James Garrett, The Elements of User Experience, 2011

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

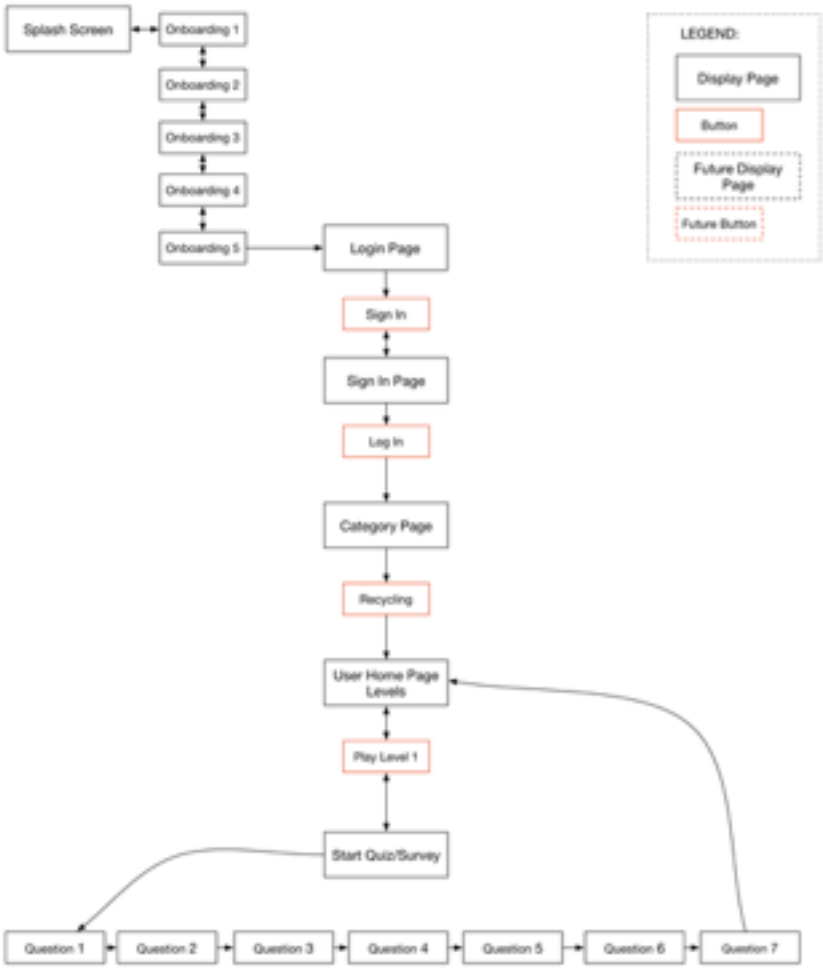
Starting Point



Rosenfeld, *The User Experience Team of One: A Research and Design Survival Guide*

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Structure



http://www.alysegilbert.com/one-drop-interactive/

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Sketching

Initial drafts of the UI should focus on content: the scope of what to include on a screen, and how to lay it out.

http://murdochcarpenter.com/portfolio/wireframes/

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

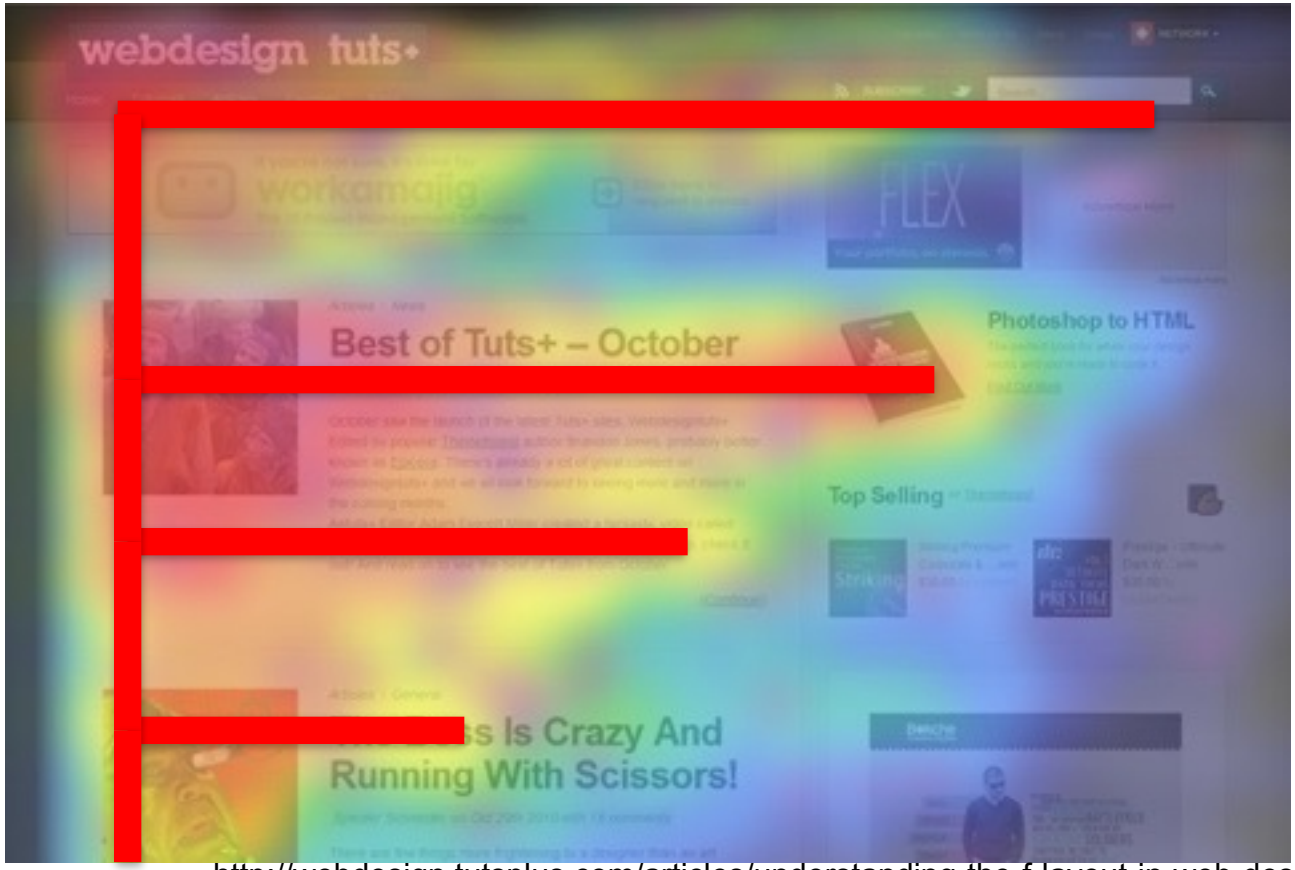
Wireframing

A wireframe is a visual representation or mock-up of a user interface, using only simple shapes

- focuses on content, information hierarchy and flow
- defers details about presentation (colour, fonts, images)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

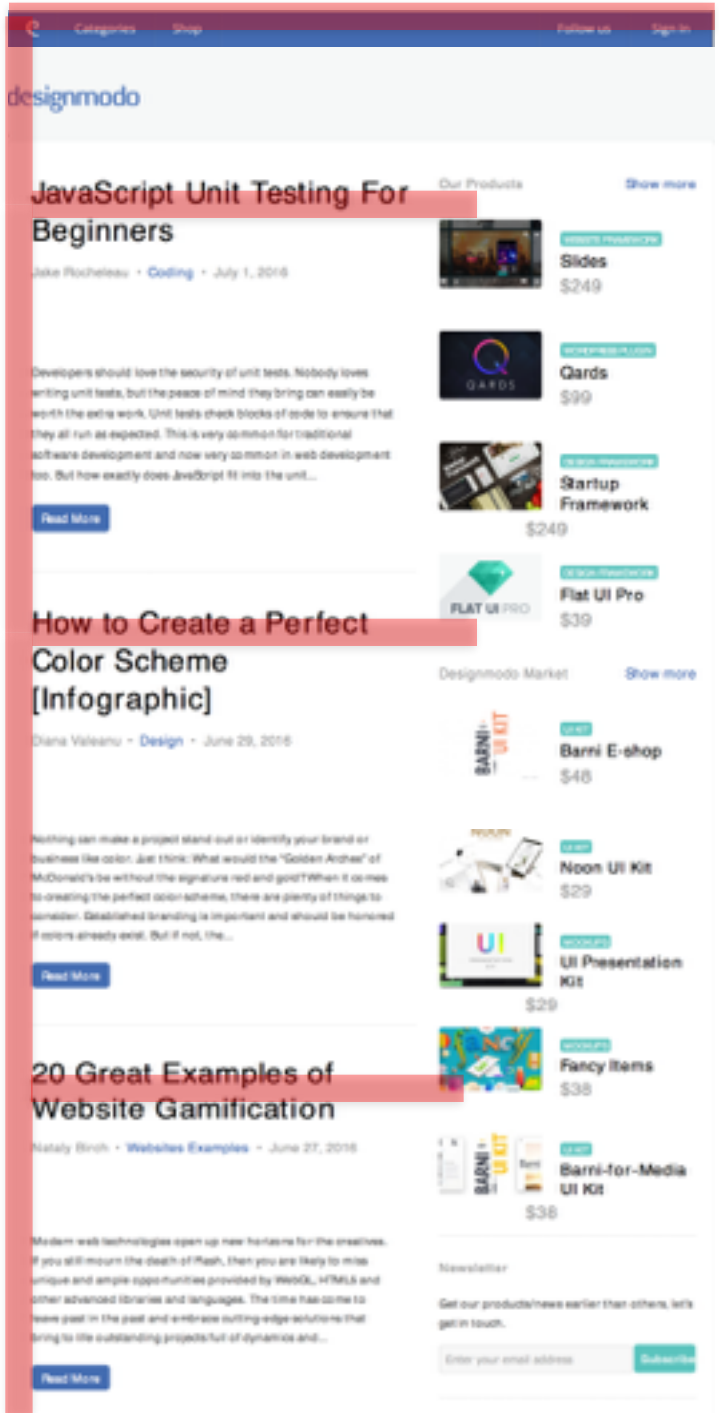
F-Layout Design

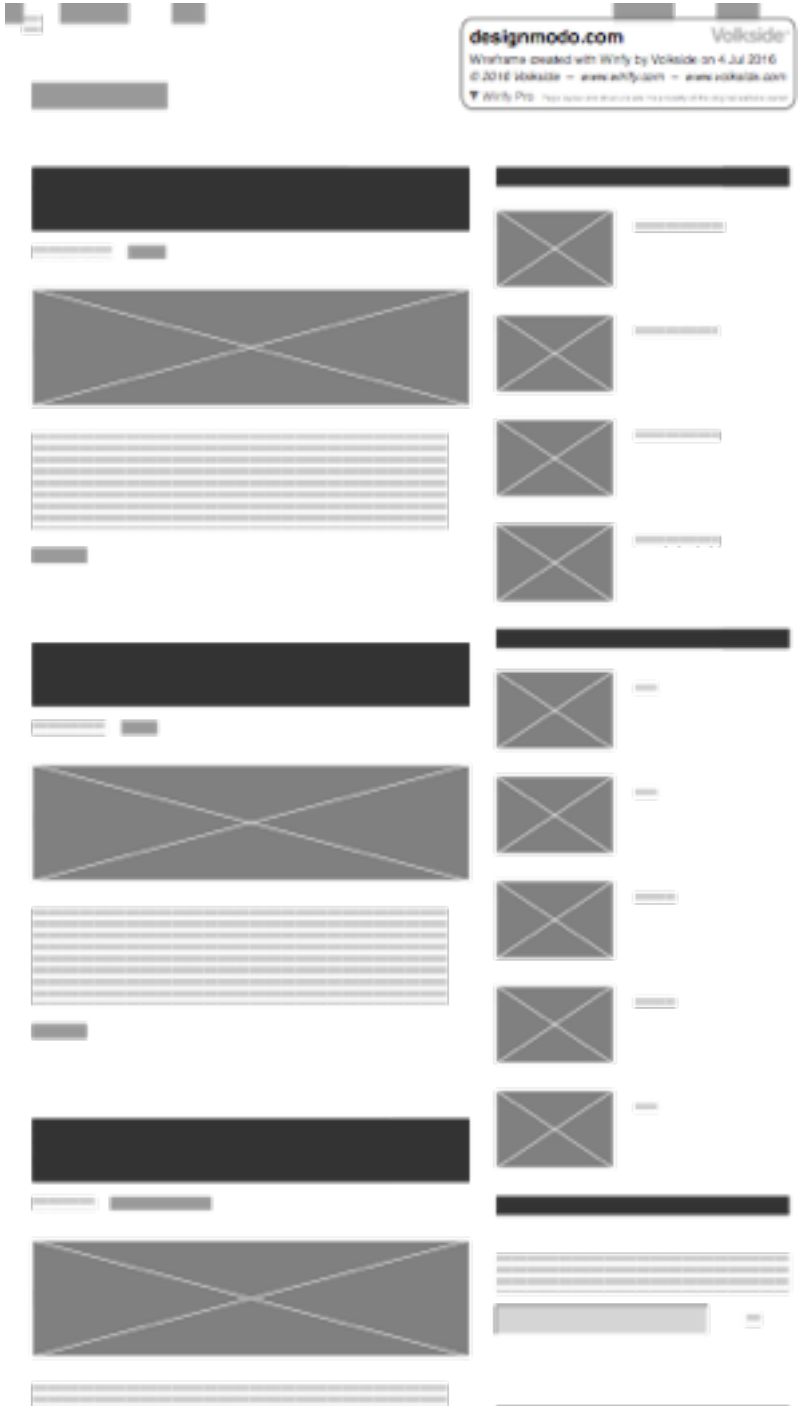


http://webdesign.tutsplus.com/articles/understanding-the-f-layout-in-web-design--webdesign-687

The heatmap shows where users tend to focus their gaze, reading left to right, and top to bottom (but less to the right the lower they gaze)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

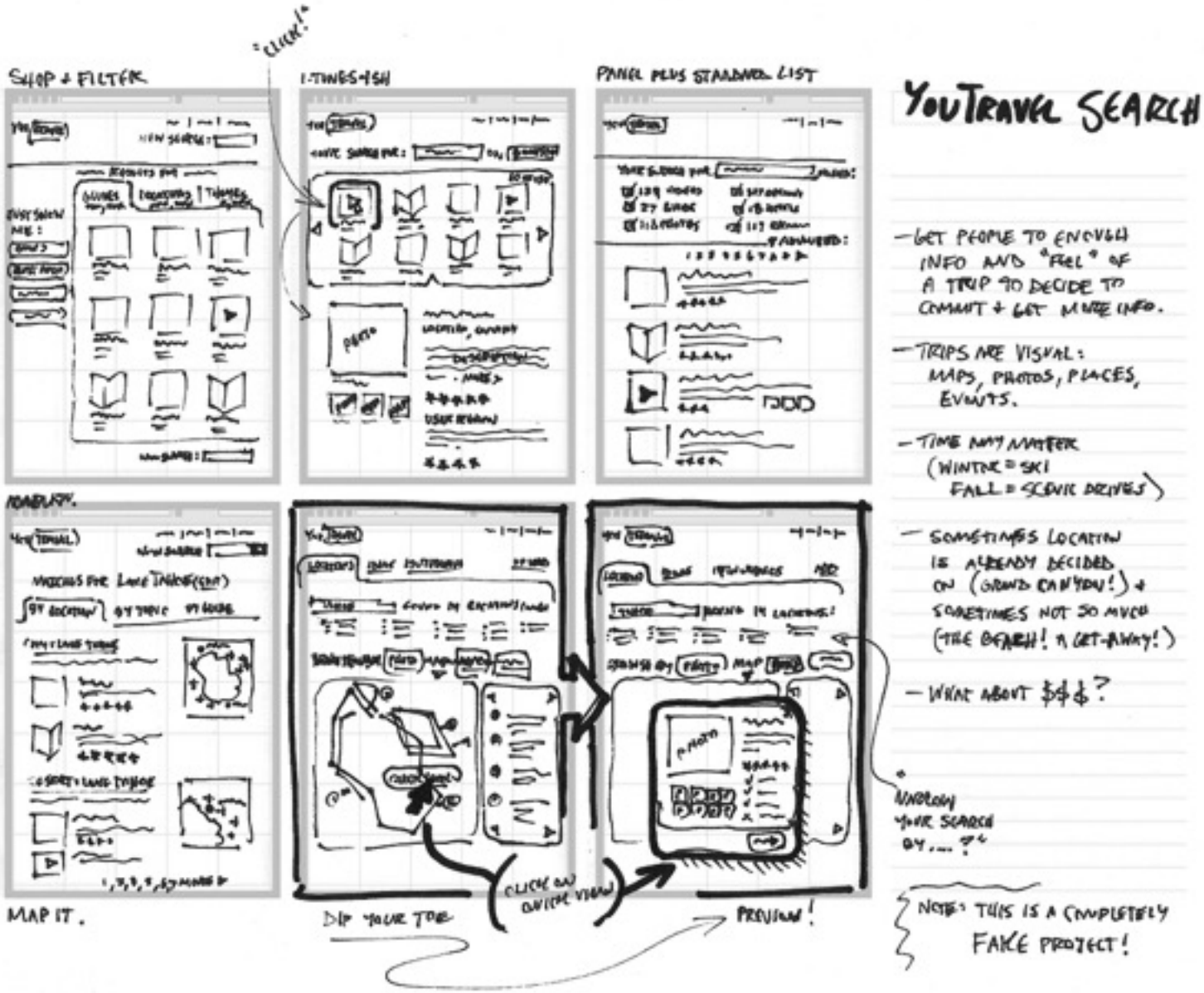

Categories
Shop
Follow us
Sign in
designmodo
JavaScript Unit Testing For Beginners
Read More
How to Create a Perfect Color Scheme [Infographic]
Read More
20 Great Examples of Website Gamification
Read More
Our Products
Show more
Slides
$249
Qards
$99
Startup Framework
$249
Flat UI Pro
$39
Designmodo Market
Show more
Barni E-shop
$48
Noon UI Kit
$29
UI Presentation Kit
$29
Fancy Items
$38
Barni-for-Media UI Kit
$38
Newsletter


designmodo.com
Volkside

Sketching vs. Low-Fid Wireframing

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Sketching


YOUTRAVEL SEARCH
SHOP + FILTER
ITINERARY
PANEL PLUS STANDARD LIST
ROADTRIP.
MAP IT.
DIP YOUR TOE
PREVIEW!
CLICK ON OVERVIEW
NOTE: THIS IS A COMPLETELY FAKE PROJECT!

Iterative Process

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Sketchboard

Alternatives

User Flow, Storyboard

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Sharing the Sketchboard

- Walk the team through the sketchboard, and present the sketches as potential starting points for solutions
- Can roll up the sketchboard and take it to clients and partners for walkthroughs, feedback, and suggestions for improvement
- Give reviewers pens to write comments
  - opinions, strengths, weaknesses, suggestions

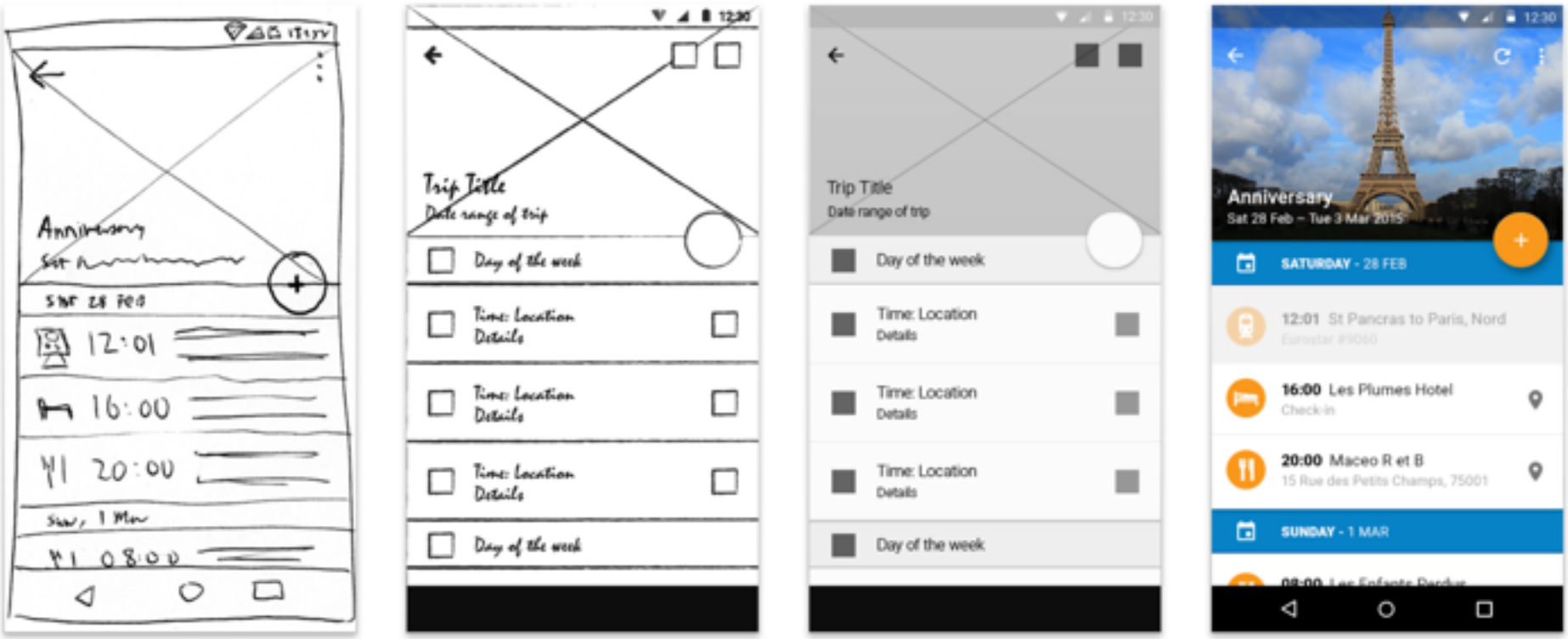
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Apple's Design Process

- **Pixel Perfect Mockups**
- **10 to 3 to 1**. Apple designers start with 10 mockups for any product or feature, which they whittle to 3, refine those into one design
- **Paired Design Meetings**. Designers have two meetings each week.
  - *Crazy brainstorming*, where they let their imaginations run wild, with no constraints
  - *Production meeting*, work out how a crazy idea from the first meeting might work.
- **Designers have complete freedom.**

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Sketching, Wireframing, Prototypes



http://murdochcarpenter.com/portfolio/wireframes/

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Mockups

A mockup sketches the essence of a solution, in terms of screen contents and navigation among screens.

A paper mockup includes

- Sketches or wireframes of screen shots
- Descriptions of each UI widget
  - purpose of widget
  - effect of input event
  - how output events affect display screens
- Navigation diagram that shows how to navigate among the screens

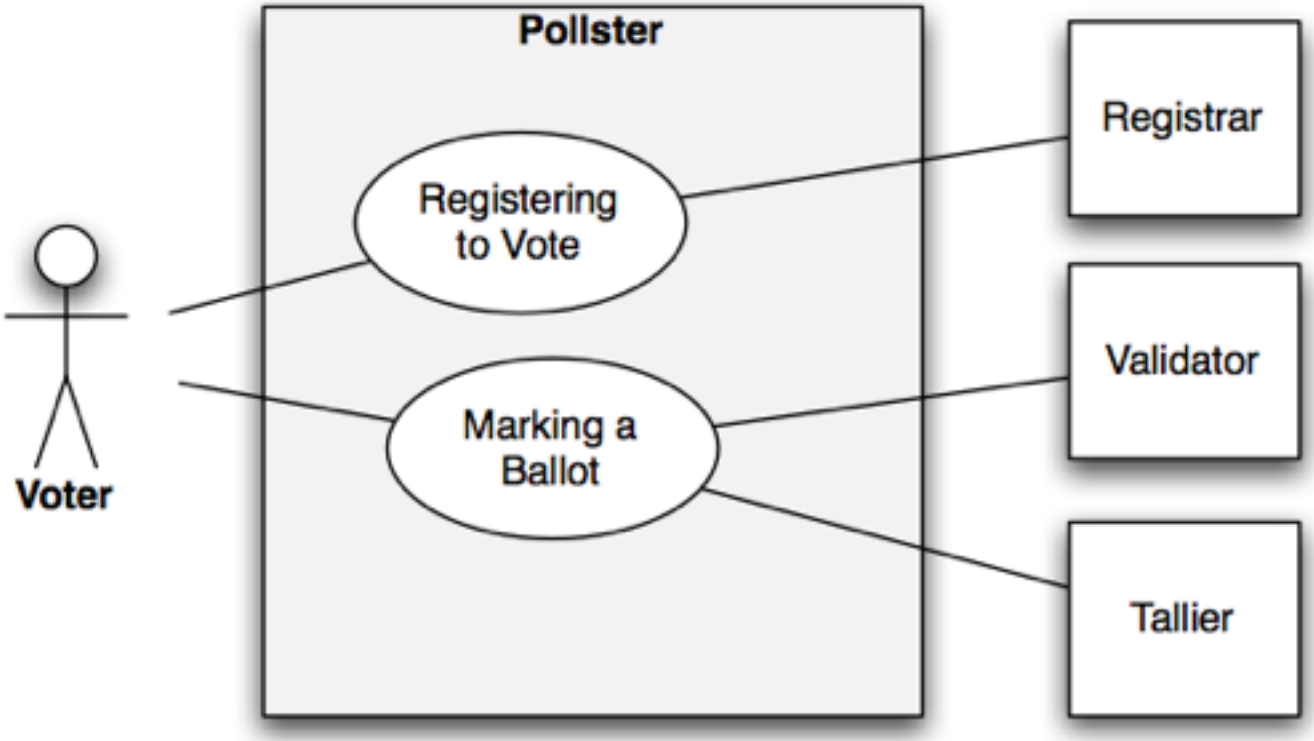
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Electronic Voting

- Registrar: The Registrar registers voters prior to an election.
- Validator: The Validator ensures that only registered voters can vote, and that only one ballot is counted for each registered voter.
- Tallier: The Tallier tallies the results of the election or survey.
- Pollster: The Pollster acts as a voter's agent, assisting with voter registration, presenting human readable ballots to a voter, collecting the voter's responses to ballot questions, performing cryptographic functions on the voter's behalf, obtaining necessary validations and receipts, and delivering ballots to the ballot box.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Use Cases



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

UC "Registering to Vote"

Precondition: The voter has obtained a voter [identification number, token, and registration address] from the election administrators.

Voter «actor»	Pollster	Registrar «actor»
1. Voter invokes the *sensus* command, to run the Pollster.		
	2. Pollster displays a menu with options "register to vote" and "mark ballot". 1	
3. Voter selects the option "register to vote".		
	4. Pollster generates and shows a public/private key pair for Voter. 2	
5. Voter hits "ok" button.		
	6. Pollster prompts voter for his/her [identification number, token, and registration address]. 3	

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

UC "Registering to Vote"

Voter «actor»	Pollster	Registrar «actor»
7. Voter types his/her [identification number, token, and registration address].		
8. Voter hits "ok" button.		
	9. Pollster prepares registration request on Voter's behalf.	
	10. Pollster submits registration request to registrar.	
		11. Registrar sends acknowledgement to Pollster within a few seconds.
	12. Pollster prompts Voter for a file name for saving his/her registration information. 4	
13. Voter types in a filename he/she will remember to Pollster; this name must be unique for the registration authority.		

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

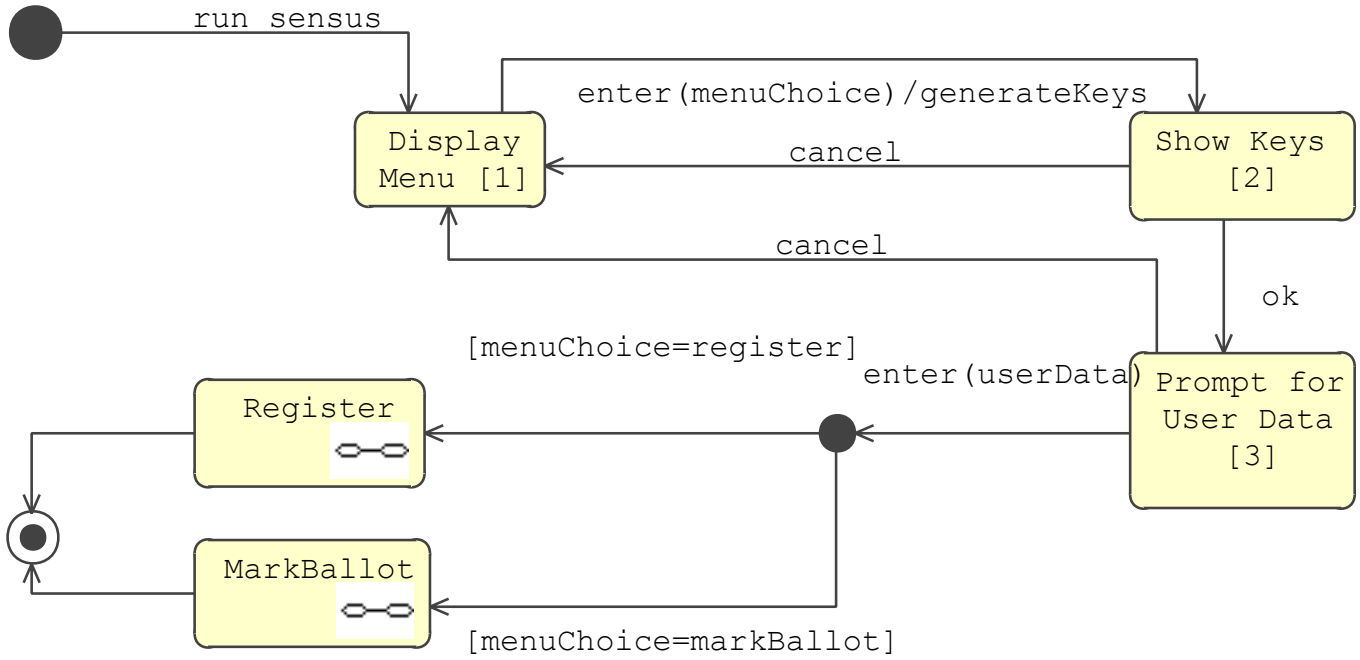
Annotating State Machines

State machines can serve as an executable navigation diagram

- Inputs: Transitions are triggered by textual input, widget selection, or mouse clicks
- Outputs: UI screen shots are displayed as state or transition actions.

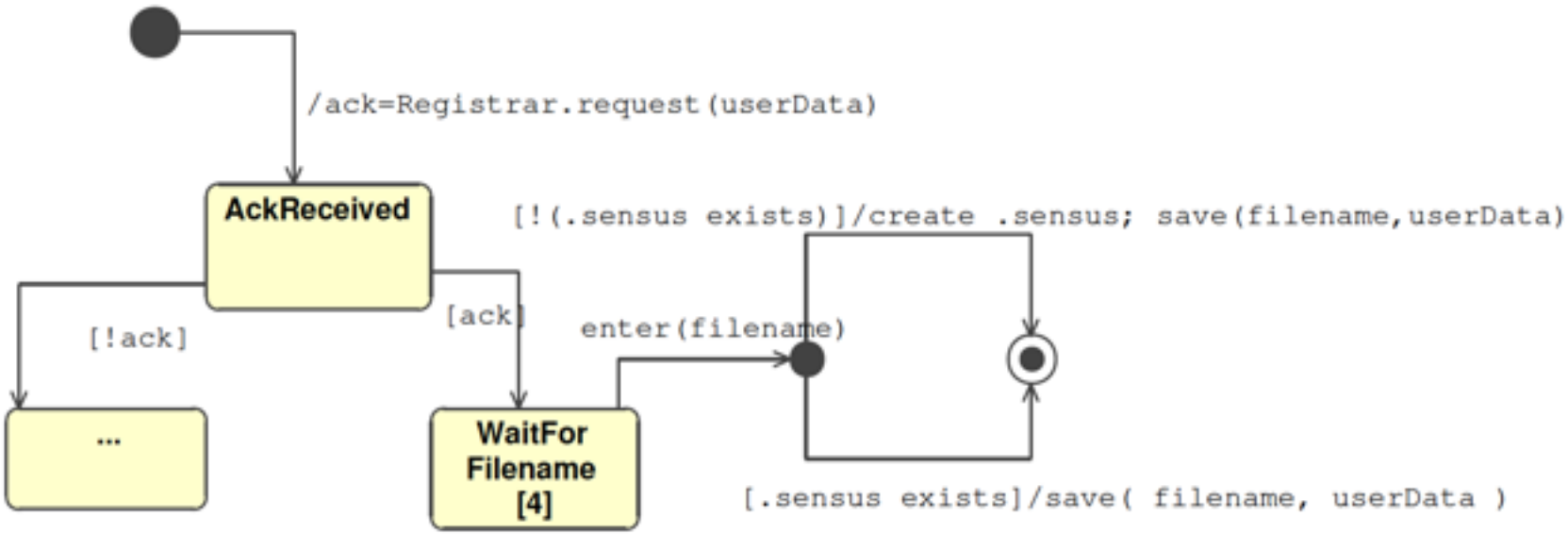
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

State Machine



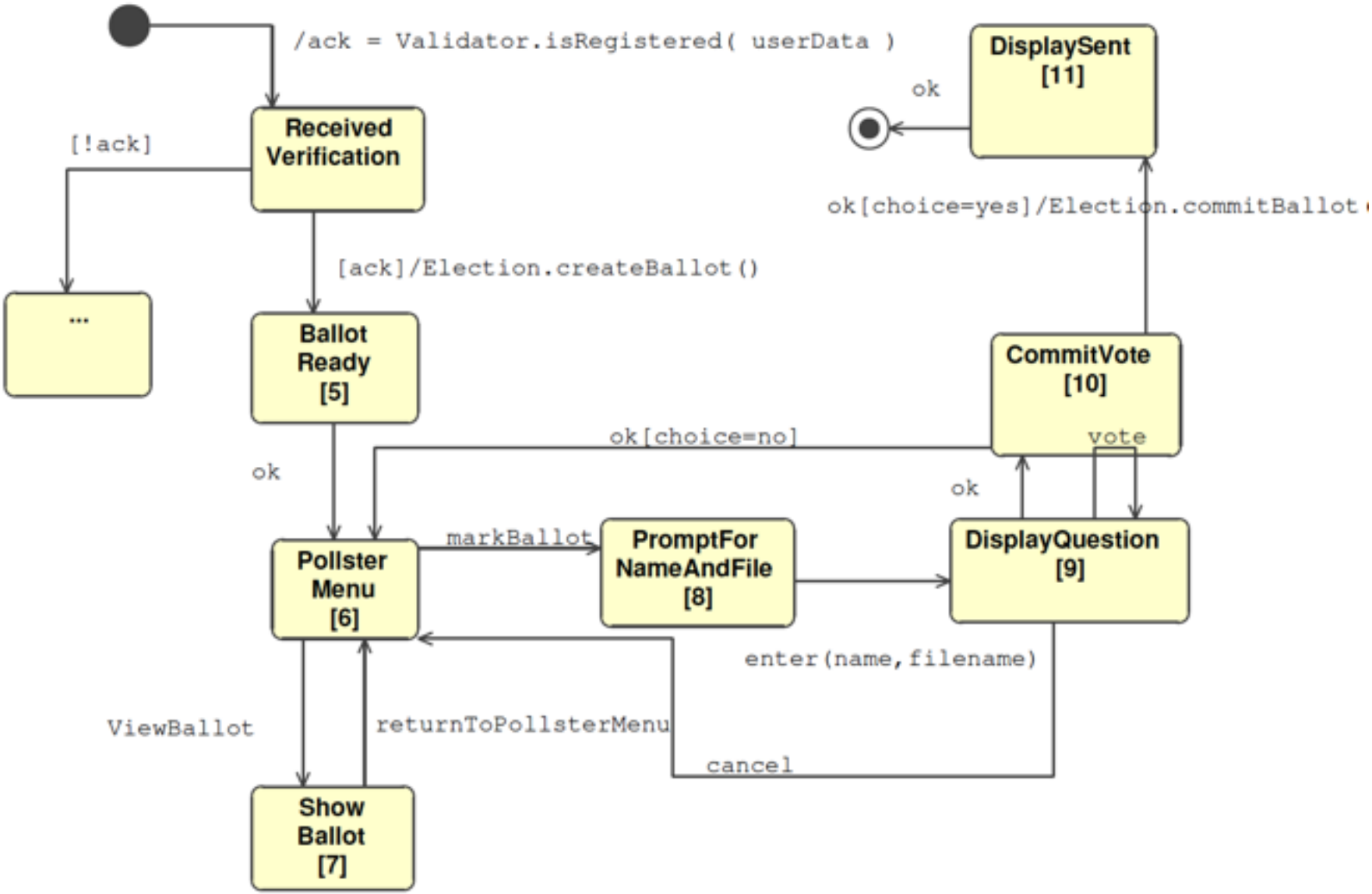
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

“Register” Sub Machine



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

“Mark Ballot” Sub Machine



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Screens

Wireframes of the screens follow this slide.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Menu of Options 1

Select one option:

- [ ] Register to vote
- [ ] Mark ballot

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Public/Private Key 2

Your public/private key pair is:

555555555555555555555555555555/
666666666666666666666666666666

Don't forget it!

OK

CANCEL

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Registration Prompt 3

Please enter your

Voter identification no.

Voter token

Registration address

OK

CANCEL

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

File Name Prompt 4

Please enter a file name for saving your registration information.

OK

CANCEL

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Send Unvoted Ballot 5

Your pollster has just been sent an unvoted ballot.

You may now choose to mark your ballot.

OK

CANCEL

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

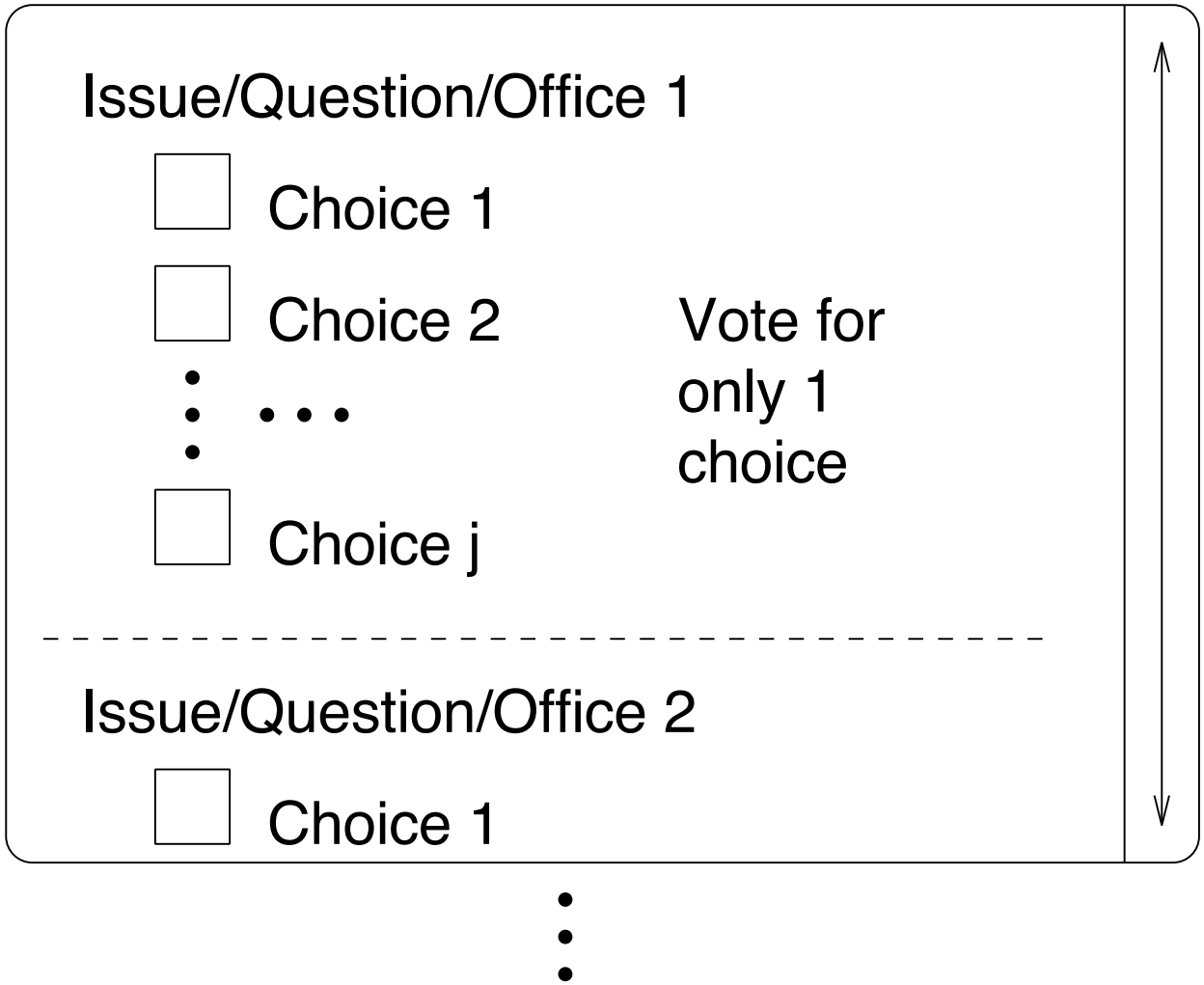
Pollster Menu 6

Select one command:

- [ ] View ballot questions and instructions
- [ ] Mark ballot

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

View Ballot 7



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Name Ballot 8

Please enter the name of the ballot for which you wish to vote.

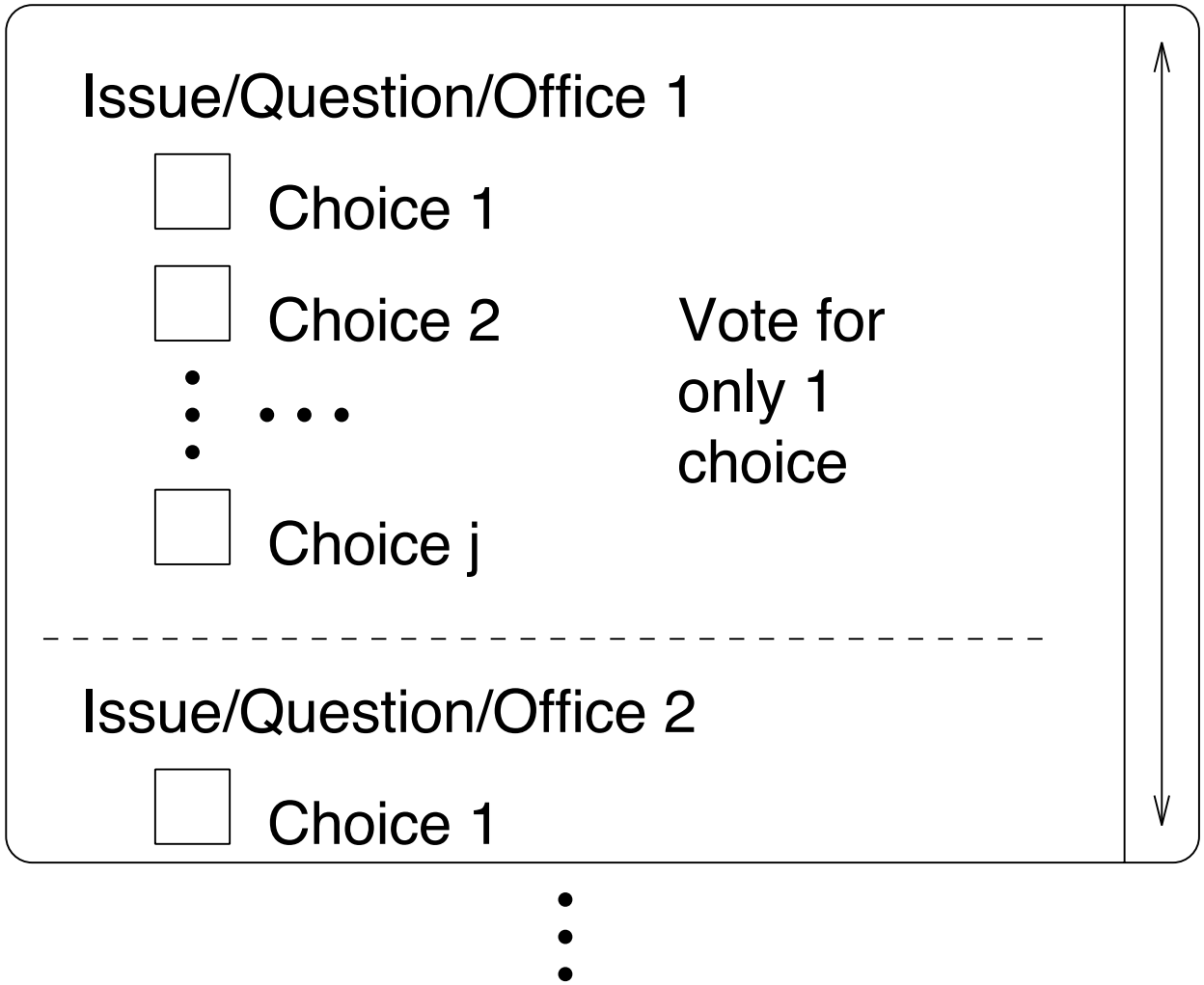
Please enter the name of your registration file, found in your .sensus directory.

OK

CANCEL

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Display Ballot Question 9



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Vote Ending Question 10

Vote ending question i

- [ ] Yes
- [ ] No

Choose exactly 1 choice

OK

CANCEL

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Voted Ballot Sent 11

Your voted ballot has been sent to the election authority.

Answering OK will exit sensus.

OK

CANCEL

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Annotating Screens

- Annotate each UI widget (e.g., button, menu option)
- Describe the purpose of each UI widget
  - purpose of widget
  - effect of input event
- Describe effect of output events on window displays (1-2 line description)
- Describe how to navigate among windows

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Screen 2

Your public/private key pair is:

❶ 555555555555555555555555555555/
❷ 666666666666666666666666666666

Don't forget it!

❸ OK ❹ CANCEL

❶ **Public key (output text):** Voter's public key

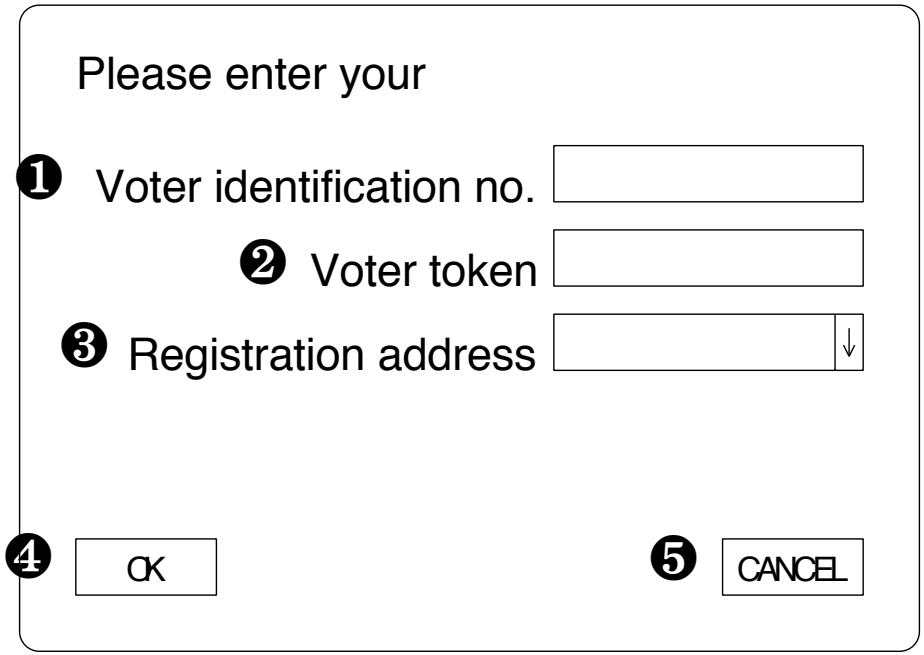
❷ **Private key (output text):** Voter's private key

❸ **Ok button (input button):** moves the system to screen shot 3

❹ **Cancel button (input button):** moves the system to screen shot 1

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Screen 3



❶ **Voter Id no. text box (input text box)**: input field for voter id no.

❷ **Voter Token text box (input text box)**: input field for voter token

❸ **Registration address text box (input menu)**: pulldown list of possible registration addresses

❹ **Ok button (input button):** moves the system to screen shot 4 (if registering to vote) or screen shot 5 (if marking ballot).

❺ **Cancel button (input button):** moves the system to screen shot 1

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

GUI Dependent Annotations

GUI-dependent: annotate models directly with GUI events

- GUI events (e.g., button pressed) to trigger behaviour
- GUI actions (e.g., Display “OK”) as system reactions

The input events and transition actions in the Electronic Voting example are GUI dependent

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

GUI Independent Annotations

GUI-independent: annotate models with macro events and actions, which map indirectly to GUI events

- Declare macros for GUI events (e.g., button pressed)
- Declare macros for GUI actions (e.g., Display “OK”)
- Annotate models with macros

The screen outputs in the Electronic Voting example are GUI independent

GUI independent models take longer to create, but are easier to maintain (because changes to the GUI do not impact the model)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

- Screen shots may be created using any method you like.
- Don't get caught up with fancy features of a UI (e.g., colours, position on screen, type of button).
  - The screen shots in a UI specification are intended to show when different inputs are required and the form of the input (e.g., textbox, button, etc.)
  - Style will be added later in the product's development.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

- User Experience (EX) aims at the ultimate (visceral) positive reaction of users to products and companies
- Sketching enables rapid generation of alternative UI designs
  - Lots of ideas to work from
  - Multiple iterations and refinements
  - Active input from all stakeholders
- Mockups depict the essence of a solution, in terms of screen contents and navigation among screens.

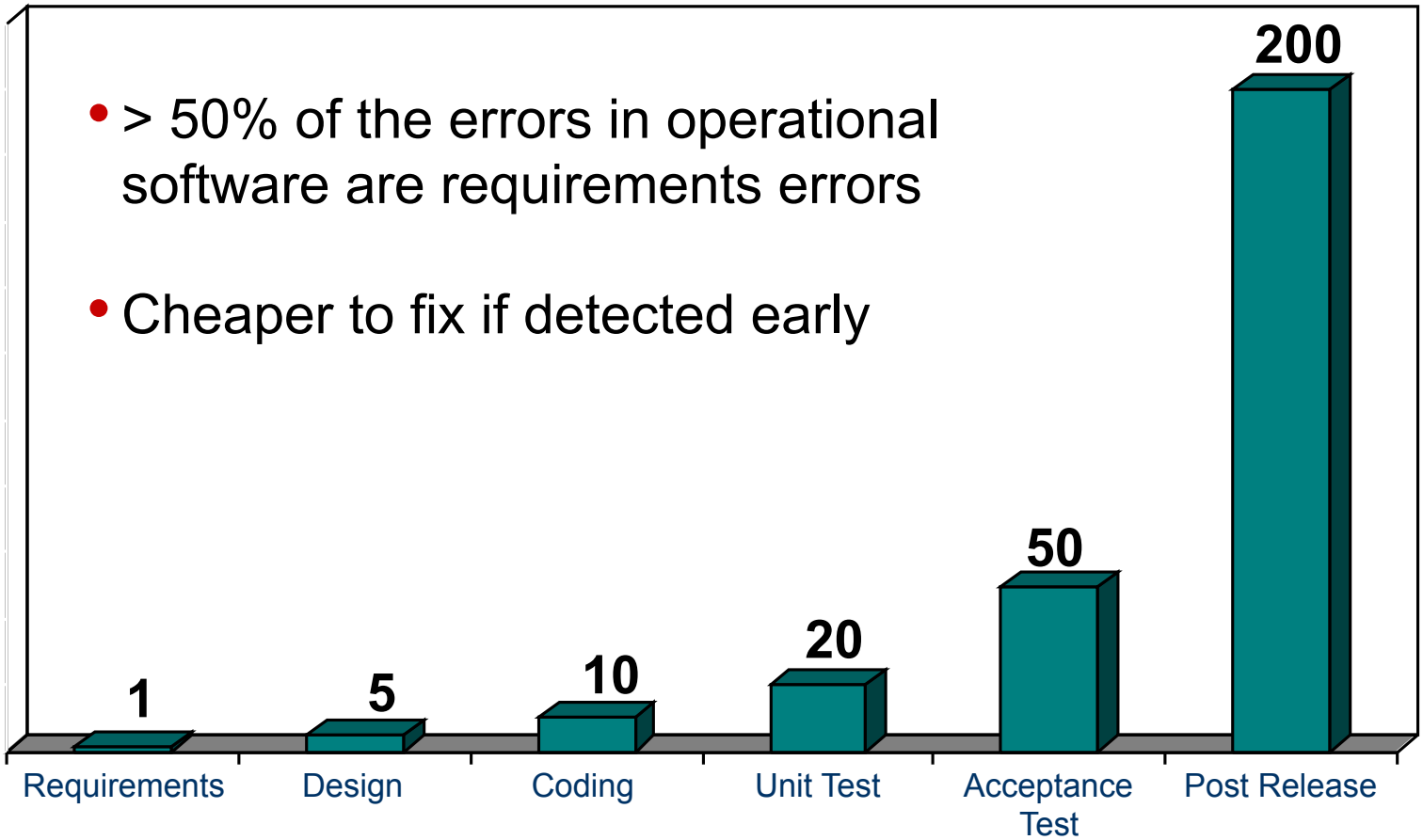
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Validation and Verification

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Recall

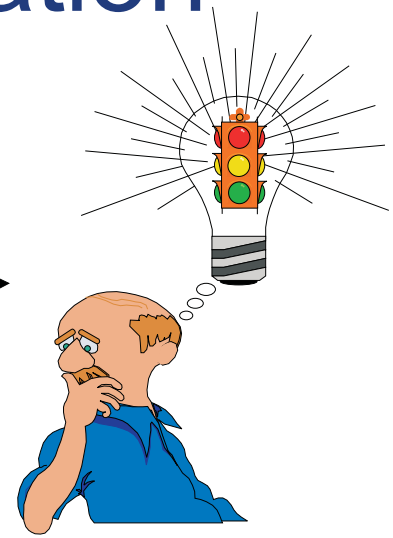


Cost to fix req error once discovered

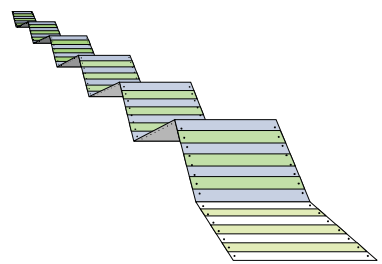
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Validation vs. Verification

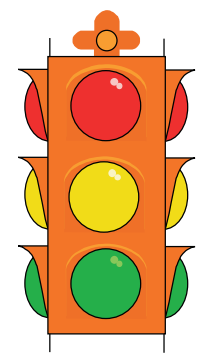
Validation:
Are we building the right product?



Requirements Specification



Verification:
Are we building the product right?



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Validation Goals

We want to ensure that

- The specification captures the stakeholders' requirements
- The customer/user is satisfied that the system as specified will fit their needs
- The developers understand the specification

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Validation Criteria

- Well-formed
  - unambiguous
  - consistent
  - ranked for importance and stability
  - modifiable (easy to change specification)
  - verifiable
- Valid
  - correct
  - complete

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Reviews

- Humans (often semi-outsiders) read and analyze artifacts, look for problems, meet to discuss these problems, and agree on a set of actions to address the identified problems.

- Reviews work.
  - They find more errors than testing does.
  - They find errors faster than testing does.
  - Everyone believes in them

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Reviews

There are a number of techniques for reviewing specifications.

- Peer reviews
  - Quota on positive and negative comments
  - Signing Off
- Group Reviews:
  - Walkthroughs
  - Focused Reviews
  - Formal Inspections
  - “Active Reviews”

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Walkthroughs

Expert or author presents the specification (scenarios, models, sketchboards, storyboards), and the audience reviews the work from the presentation.

Advantages:

- no prep work, so reviewers may be more likely to attend than if they had to read the document in order to participate
- may be the only way get feedback from busy stakeholders (e.g., customers)
- if the requirements / specifications are presented to a large audience, representing a broad cross section of skills and viewpoints, then there is a hope that major oversights will be caught

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Focused Reviews

The reviewing task is decomposed into subtasks, where reviewers have roles and each looks only for specific types of errors (e.g., per use case, or per error type)

Advantages:

- avoids the problem of reviewers not having the time to read the whole document
- each reviewer is more effective because he or she can focus on a handful of error types
- the leader can assign each reviewer to tasks for which he or she is the most skilled

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Formal Inspections

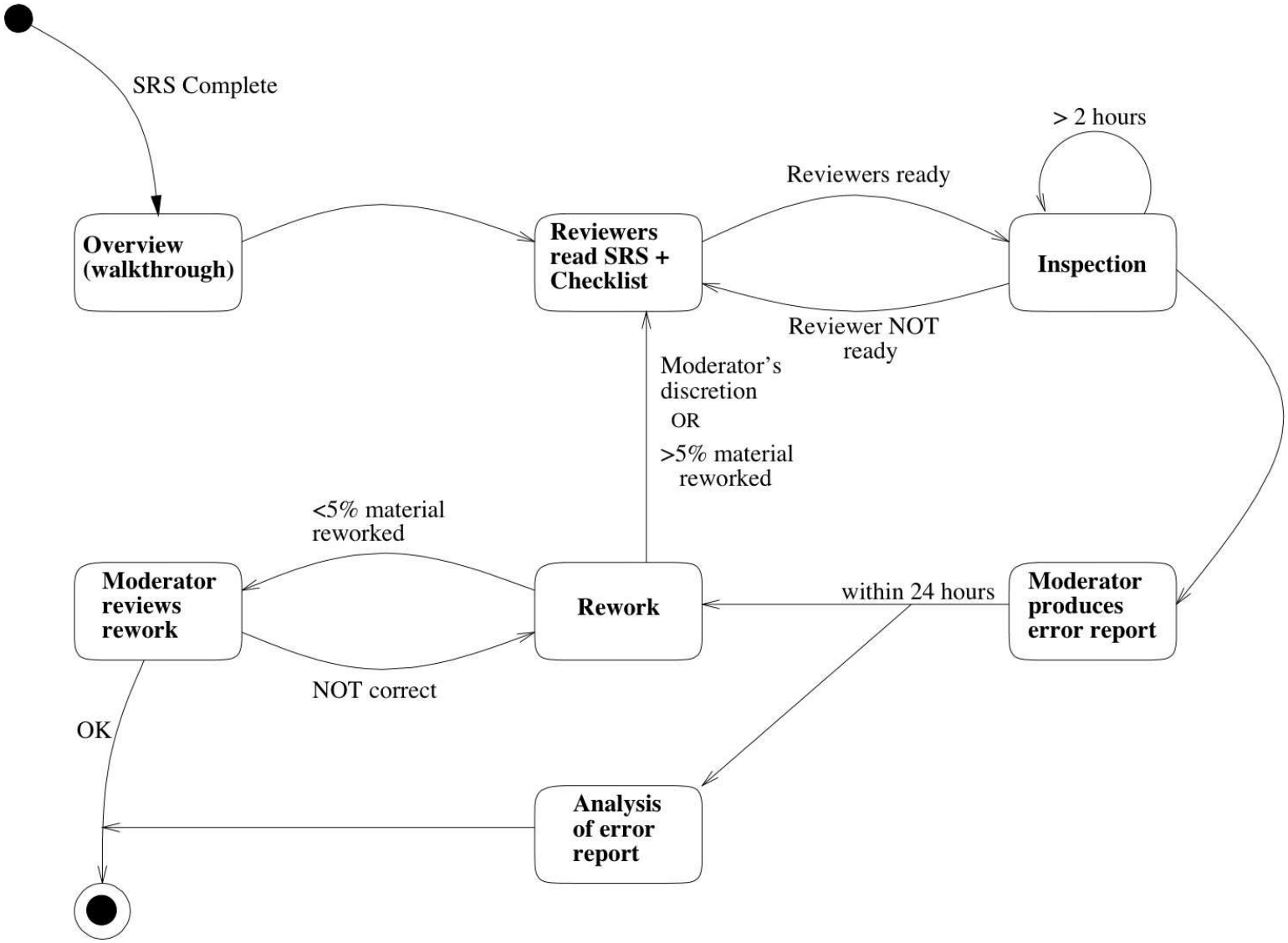
Extreme inspection!

A formal inspection is a managed review process

- with rules concerning participants and roles
- strict entry and exit criteria for each step in process.
- primary goal is to improve the quality of the SRS.
- secondary goal is to improve the quality of the development process

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Formal Inspections



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Formal Inspections

What can an analysis of detected errors tell us?

- It can reveal new types of errors that should be added to the checklists to help with future inspections.
- It can point to projects or use cases that are likely to be problematic, because significantly more errors were reported than usual.
- Evaluation of entry and exit points can help determine whether the project is on schedule.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

NASA

NASA requires peer reviews and inspections to be performed on requirements, designs, code, and test plans for projects developing the following software classes:

- **Human Rated Space Software Systems**
- **Non-Human Space Rated Software Systems or Large Scale Aeronautics Vehicles**
- **Mission Support Software or Aeronautic Vehicles, or Major Engineering/Research Facility Software**
- **Basic Science/Engineering Design and Research and Technology Software** (if safety-critical)

It is up to the program/project/facility manager to determine the type of inspections or peer reviews to conduct and on which software products. Software Formal Inspections are recommended for the most critical products and should be performed early, on models, requirements, and preliminary design products, to provide the greatest return on finding and correcting errors and defects before they are propagated into detailed design, code and operations.

NASA, *Software Engineering Requirements*, NPR 7150.2B, 2014

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example Checklist (NASA)

System Requirements Inspection

- Describe proper allocation of functions to software, firmware, hardware, and operations.
- Address the validation of all external usage interfaces.
- Check that all the software system functions are identified and broken into configuration items, and that the boundary between components is well-defined.
- Check that all configuration items within the software system are identified.
- Check that the identified configuration items provide all functions required of them.
- Check that all interfaces between configuration items within the software system are identified.
- Address the correctness of the software system structure.
- Check that all quantifiable requirements and requirement attributes have been specified.
- Address the verifiability of the requirements.
- Check for the traceability of requirements from mission needs (e.g., use cases, etc).
- Check for the traceability of requirements from system safety and reliability analyses (e.g., Preliminary Hazard Analysis (PHA), Fault Tree Analysis (FTA), Failure Mode and Effects Analysis (FMEA), hazard reports, etc.).

NASA, *Software Formal Inspection Standard*, NASA-STD-8739.9, 2013

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example Checklist (NASA)

Software Requirements Inspection

- Check that the specification of each of the following is complete and accurate:
  - Software functions.
  - Input and output parameters.
  - States and modes.
  - Timing and sizing requirements for performance.
  - Interfaces.
  - Use Cases if available.
- Check that specifications are included for error detection and recovery, reliability, maintainability, performance, safety, and accuracy.
- Check that safety requirements are identified as such.
- Check that safety-critical modes and states, and any safety-related constraints, are identified.
- Address the traceability of requirements from higher level documents.
- Check that the requirements provide a sufficient base for the software design.
- Check that the requirements are measurable, consistent, complete, clear, concise, and testable.
- Check that the content of the software requirement specification fulfills the NPR 7150.2 (NPR 7150.2 revision A requirement SWE-109).

NASA, *Software Formal Inspection Standard*, NASA-STD-8739.9, 2013

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

“Active Review”

Inspection process where reviewers (who are often outsiders) act as users of our specification artifacts.

- Testing team can design tests based on the specification artifacts.
- Technical writers can write the user’s guide or help pages, based on the specification artifacts.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Reviews

Advantages

- More effective than testing (code)
  - Finds causes of errors
  - Reduces errors by factor of 5-10
  - Improves productivity by 14%-25%
  - Reduces V&V costs by 50%-80%
- Simple, doable, only “costs” time and effort
- Inspires authors to improve their work
- Can apply to all sorts of software artifacts
- Improvement claims are backed up by experiment

Disadvantages

- Some find it dull work
- Requires preparation, paperwork
- Discussion can be bogged down in disagreements or details

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Testing

Can “run” an executable model and see if its executions match expected behaviour.

Advantages

- Customer can see animation of the product executing
- Checking of low-level details is usually done reliably when done by tools
- Can be done earlier in the development lifecycle than other testing

Disadvantages

- Can be labour intensive and costly
  - hand-holding of tools
  - designing of test cases
- Many notations are not executable
- Testing can be used to show the presence of errors, but not their absence (Dijkstra)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Prototyping

"A software prototype is a partial implementation constructed primarily to enable customers, users, or developers to learn more about a problem or its solution."

Alan Davis

- Presentation Prototypes
  - used for proof of concept; explaining design features; etc.
  - explain, demonstrate and inform – then throw away
- Exploratory Prototypes
  - used to determine problems, elicit needs, clarify goals, compare design options
  - informal, unstructured and thrown away.
- Breadboards or Experimental Prototypes
  - explore technical feasibility; test suitability of a technology
  - typically no user/customer involvement
- Evolutionary (e.g. "operational prototypes", "pilot systems"):
  - development seen as continuous process of adapting the system
  - "prototype" is an early deliverable, to be continually improved.

Copyright © Steve Easterbrook

Throwaway or Evolve?

Throwaway Prototyping

Purpose:
- learn more about the problem or its solution
- discard after desired knowledge is gained

Use:
- early or late

Approach:
- horizontal - build only one layer (e.g. UI)
- "quick and dirty"

Advantages:
- learning medium for better convergence
- early feedback → less cost
- successful even if it fails!

Disadvantages:
- wasted effort if reqts change rapidly
- often replaces proper documentation of the specification
- may set customers' expectations too high
- can get developed into final product

Evolutionary Prototyping

Purpose:
- learn more about the solution
- reduce risk by building parts early

Use:
- incremental; evolutionary

Approach:
- vertical – partial implementation of all layers
- designed to be extended and adapted

Advantages:
- requirements are not frozen
- return to last version if problem encountered

Disadvantages:
- early architectural choice may be poor
- can end up with complex, unstructured system that is difficult to maintain
- optimal solutions not guaranteed
- lacks control and direction

Copyright © Steve Easterbrook

Safety Analysis

Safety analysis looks for safety hazards in the system: safety-related failures, their causes, their consequences.

Software Faults

- Data sampling rate
- Data collisions
- Illegal commands
- Commands out of sequence
- Time delays, deadlines
- Multiple events
- Unsafe modes

Failures in Environment

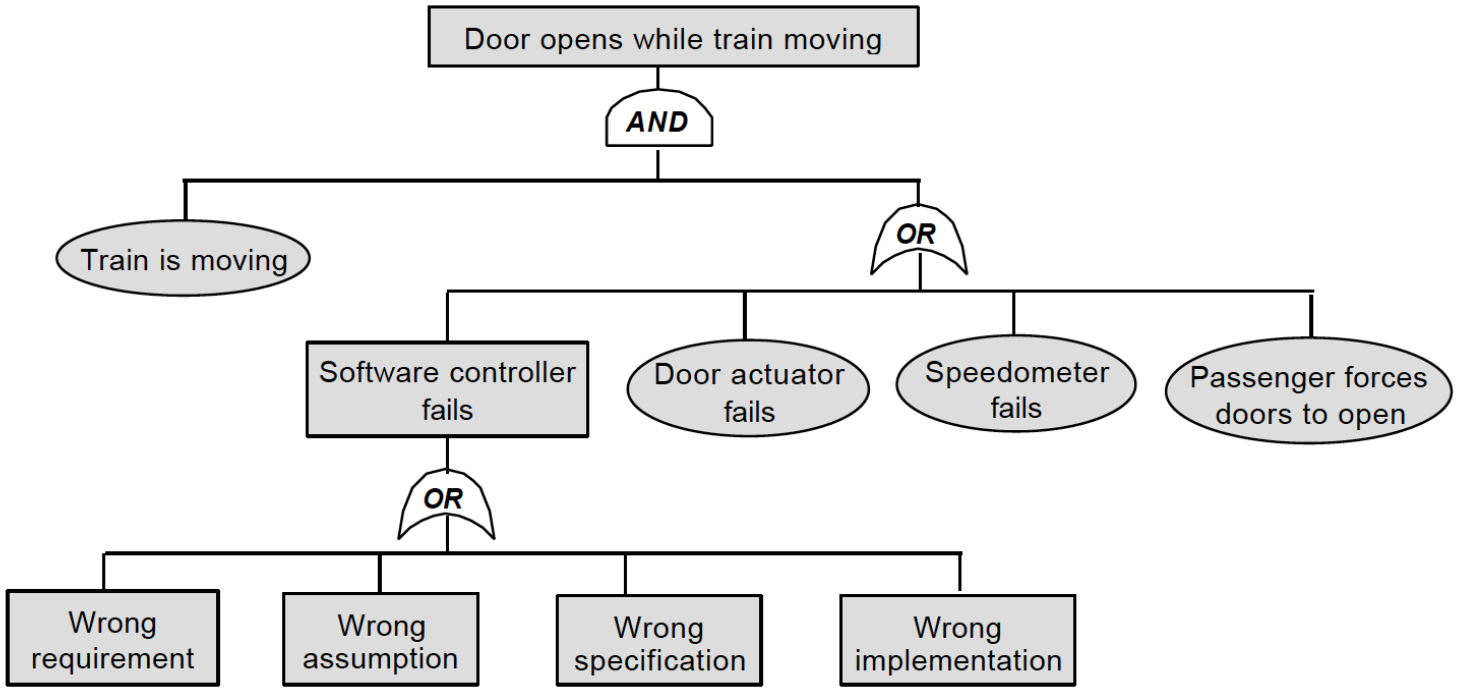
- Broken sensors
- Memory overwritten
- Missing parameters
- Parameters out of range
- Bad input
- Power fluctuations
- Gamma radiation

NASA Software Safety Guidebook

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Software Fault Tree Analysis

Software Fault Tree Analysis (SFTA) is a top-down backward-analysis approach to safety analysis that starts with an undesired software event and determines all the ways it can happen.



van Lamsveerde, *Requirements Engineering - From System Goals to UML Models to Software Specifications*, Wiley, 2007

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Independent V&V

V&V performed by a separate contractor

- Independent V&V fulfills the need for an independent technical opinion.
- Cost between 5% and 15% of development costs
- Studies show up to fivefold return on investment:
  - Errors found earlier, cheaper to fix, cheaper to re-test
  - Clearer specifications
  - Developer more likely to use best practices

Copyright © Steve Easterbrook

Summary

- Validation: check you are solving the right problem
  - Walkthroughs
  - Inspections
  - Active reviews
  - Prototypes
- Verification: check that your development steps are sound
  - Testing
  - Safety analysis
  - Consistency checking

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Dilbert on Cost Estimation

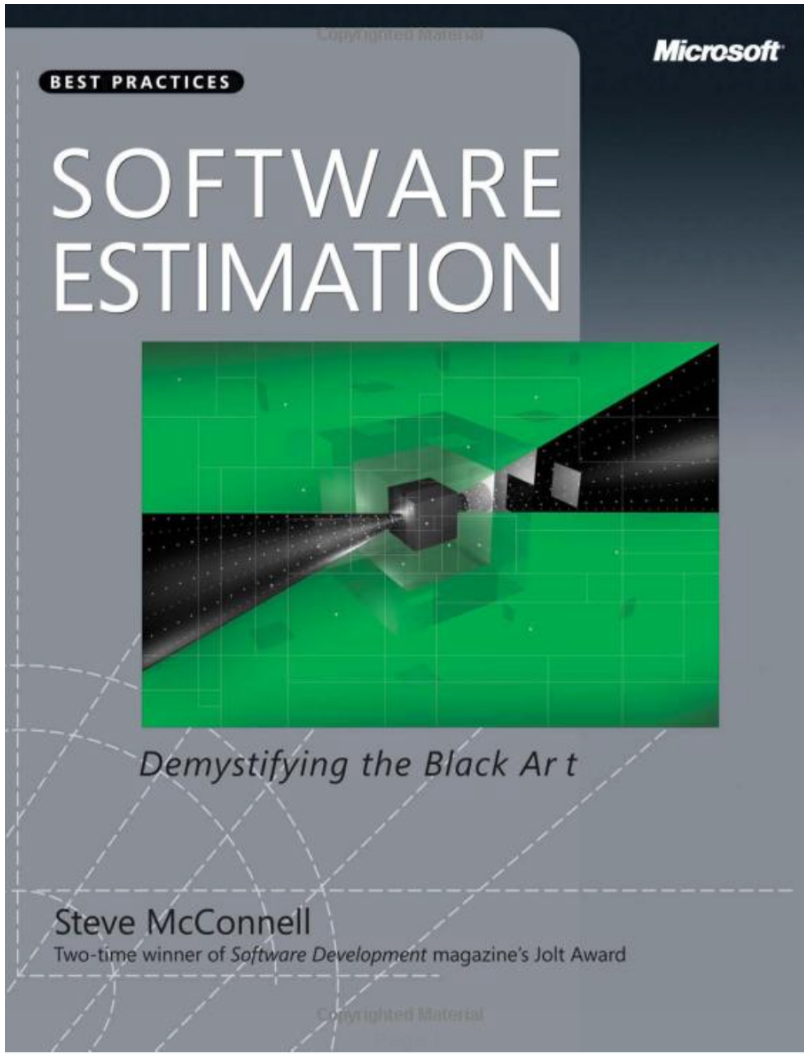
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Software Estimation

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Sources



Lecture content comes from

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Why Estimate Software Cost and Effort?

- To assess economic feasibility
- To provide a basis for agreeing to a job
- To make commitments that we can meet
- To understand resource needs

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Terminology

An estimate is a prediction of how long a project will take or how much it will cost.

A target is a statement of a desirable business objectives.

A commitment is a promise to deliver.

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Inaccurate Estimates

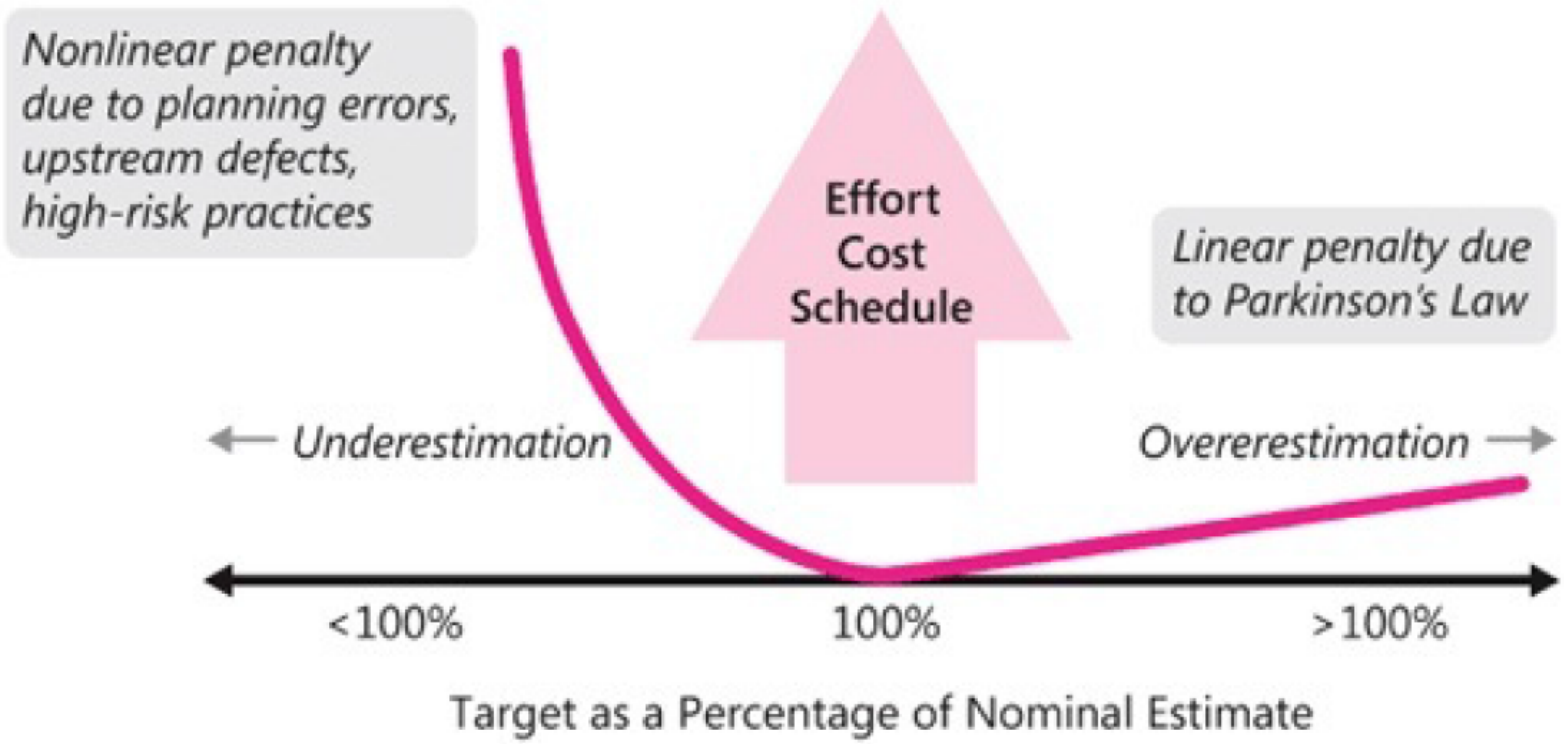
"An estimate is the most optimistic prediction that has a non-zero probability of coming true.

Accepting this definition leads irrevocably toward a method called what's-the-earliest-date-by-which-you-can't-prove-you-won't-be-finished estimating."

Tom DeMarco

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Effects of Inaccurate Estimates



Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Software Estimation

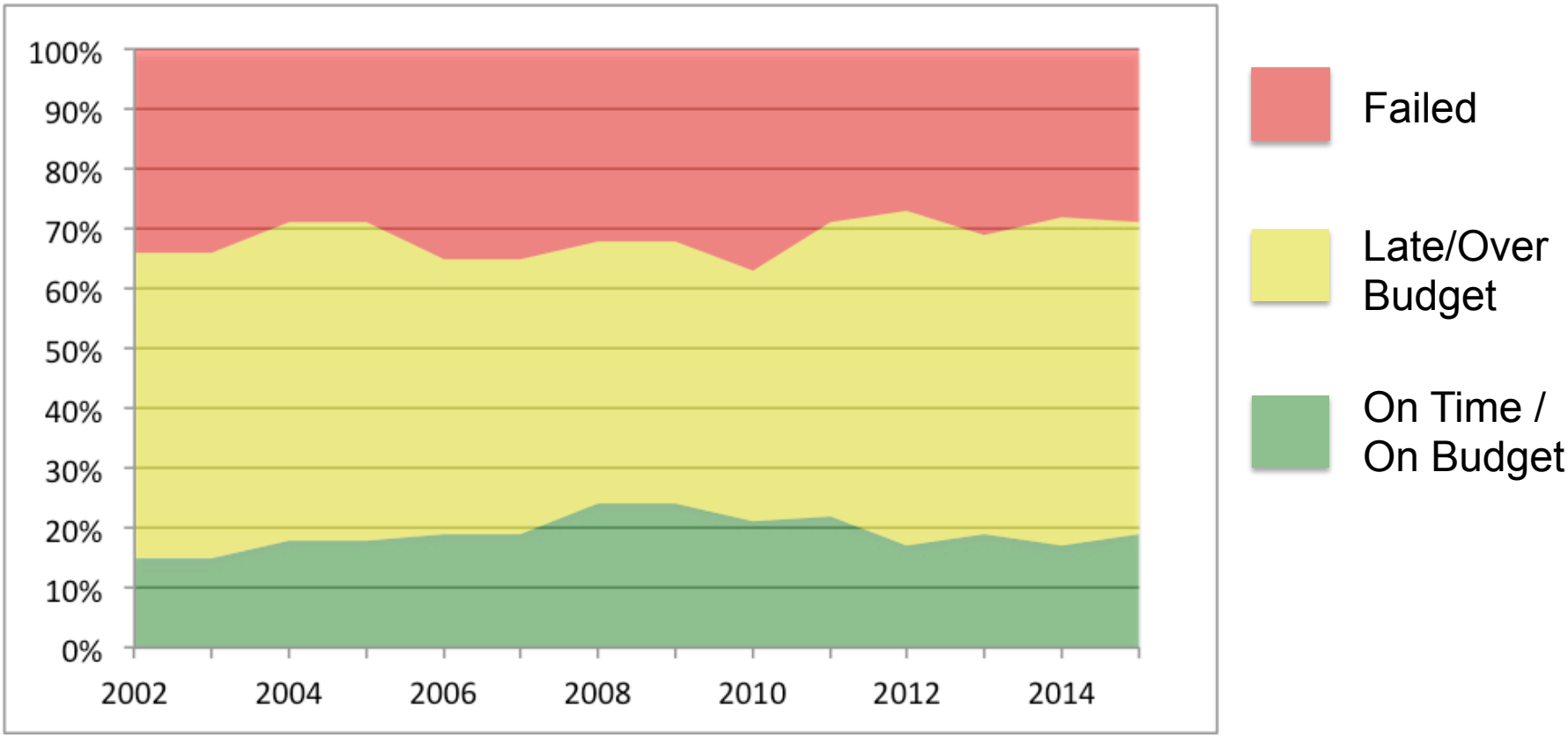
How long (hours) would it take you to implement a Stack ADT (push, pop, top)?

How long would it take you to re-implement your OS project?

How long would it take you to implement your SE463 project?

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

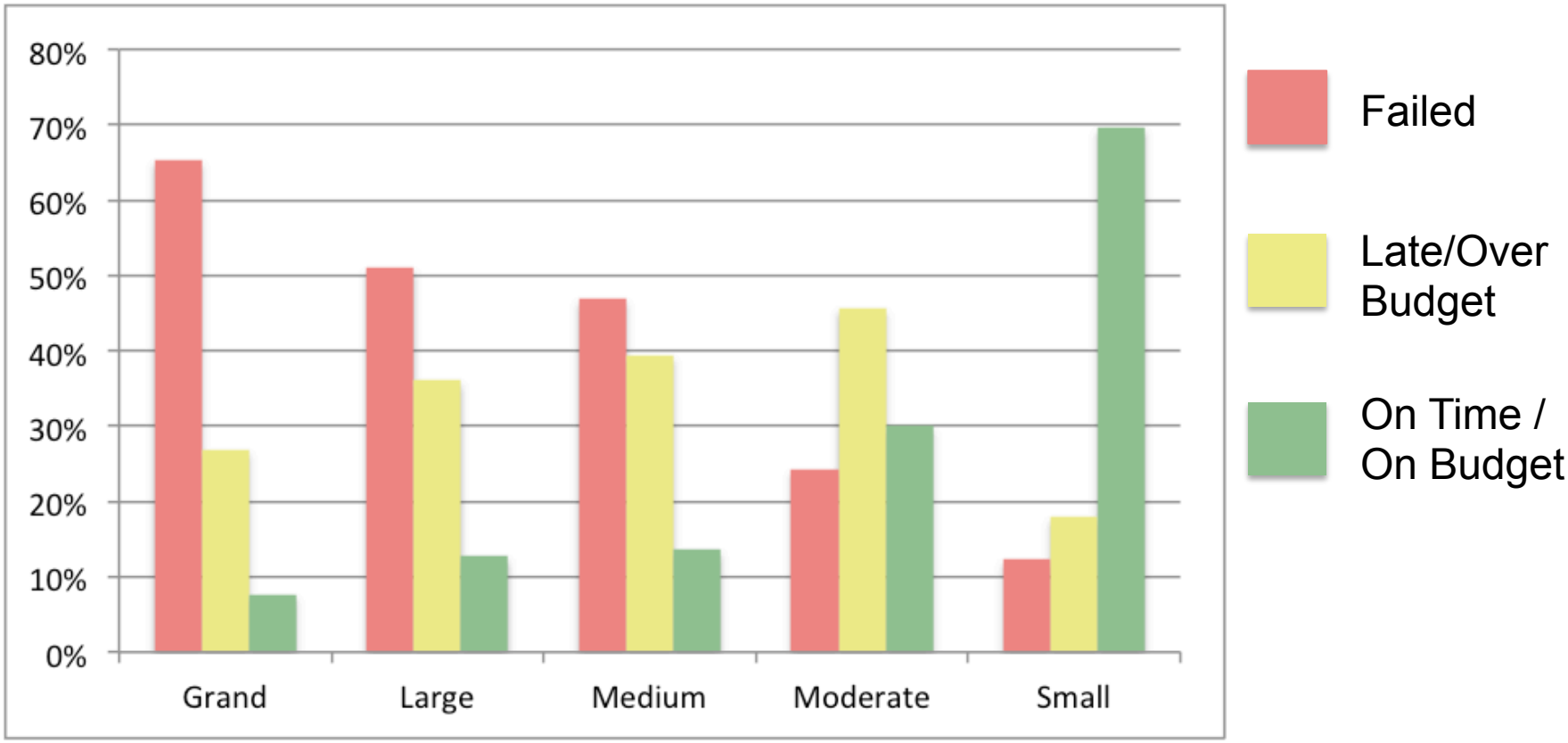
Industry's Track Record



The Standish Group, The CHAOS Report, 2002-2015

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Industry's Track Record

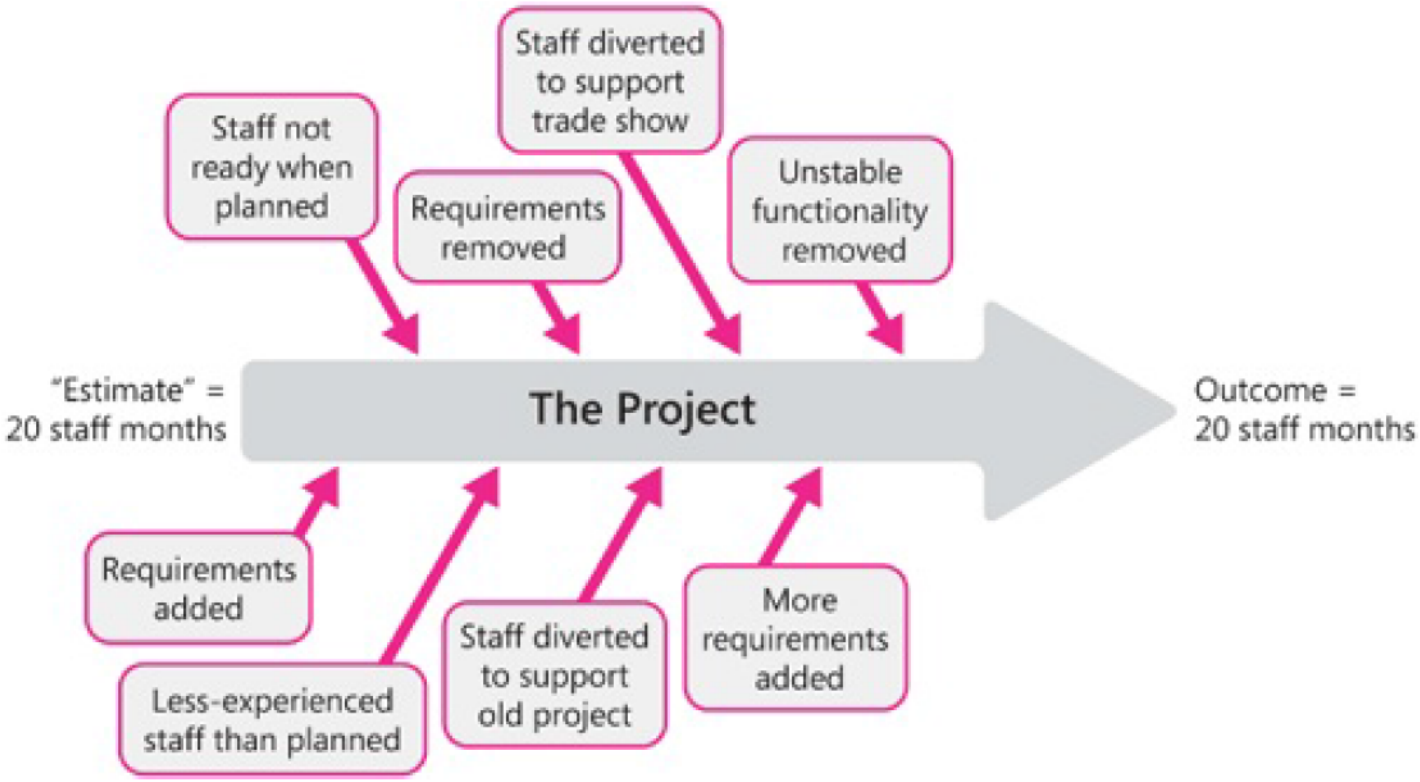


The Standish Group, The CHAOS Report, 2002-2015

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

What is a Good Estimate?

A project is successful if it is delivered on time, within budget, with acceptable functionality.



Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

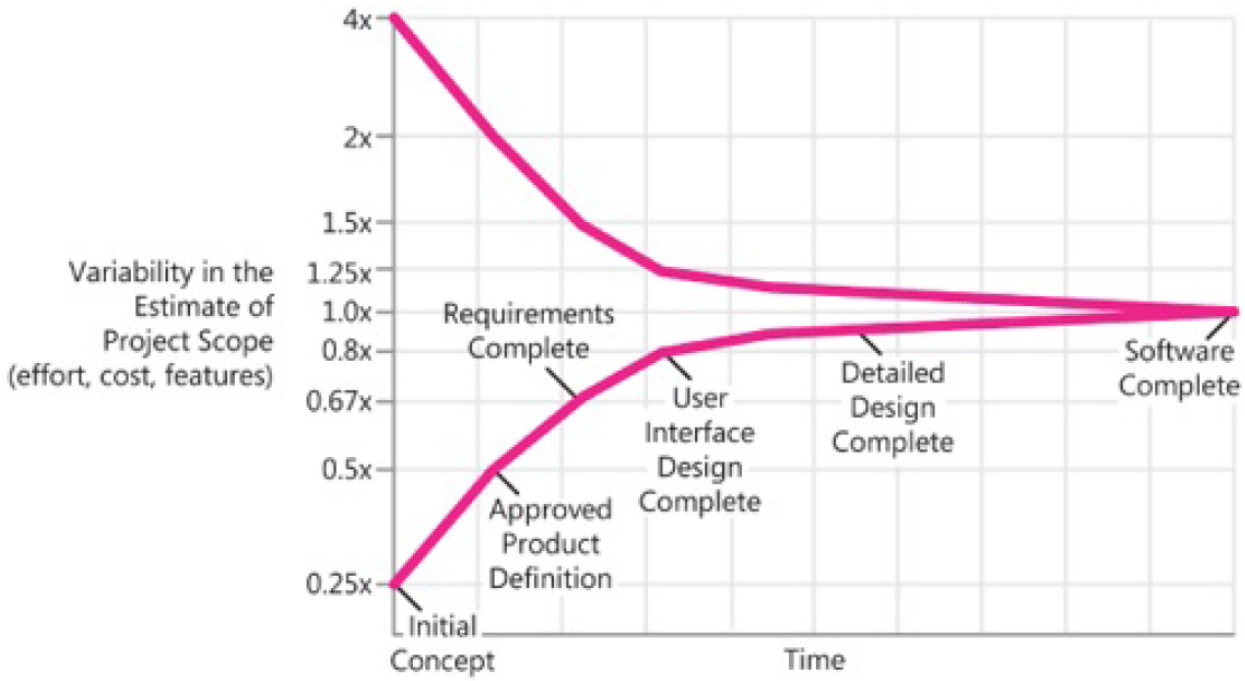
What is a Good Estimate?

Once we make an estimate, and make a commitment to deliver functionality and quality by a particular date, then we control the project to meet the target.

- remove noncritical requirements
- redefine requirements
- replacing less-experienced staff with more-experienced staff

If the initial target and initial estimate are within about 20% of each other, the project manager should have enough maneuvering room to meet the project's business goals.

Sources of Estimation Error



Uncertainty about how numerous requirements details and design decisions will be made.

Estimation accuracy improves rapidly for the first 30% of the project.

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Sources of Estimation Error

Omitted Activities

- Omitted Functional / Quality Requirements
  - Initialization, data conversion, glue code
- Software Development Activities
  - Integration, test data, performance tuning, technical reviews
- Non-Software Development Activities
  - Vacations, sick days, training, company meetings

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Sources of Estimation Error

Optimism

- Developer estimates tended to contain an optimism factor of 20% to 30%
- Managers believe that projects can be completed 30% faster than previous projects
  - Team more productive, less will go wrong, no learning curve

Bias

- Managers who want estimates to align with targets will put pressure on estimates, schedules, project teams.

Subjectivity

- Some estimation techniques include multiple adjustment knobs that allow subjectivity to creep in.

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Biggest Influences on Estimates

Project Size

- The largest driver in a software estimate is the size of the software being built
- Software projects have diseconomies of scale: larger projects require more coordination and communication

Kind of Software

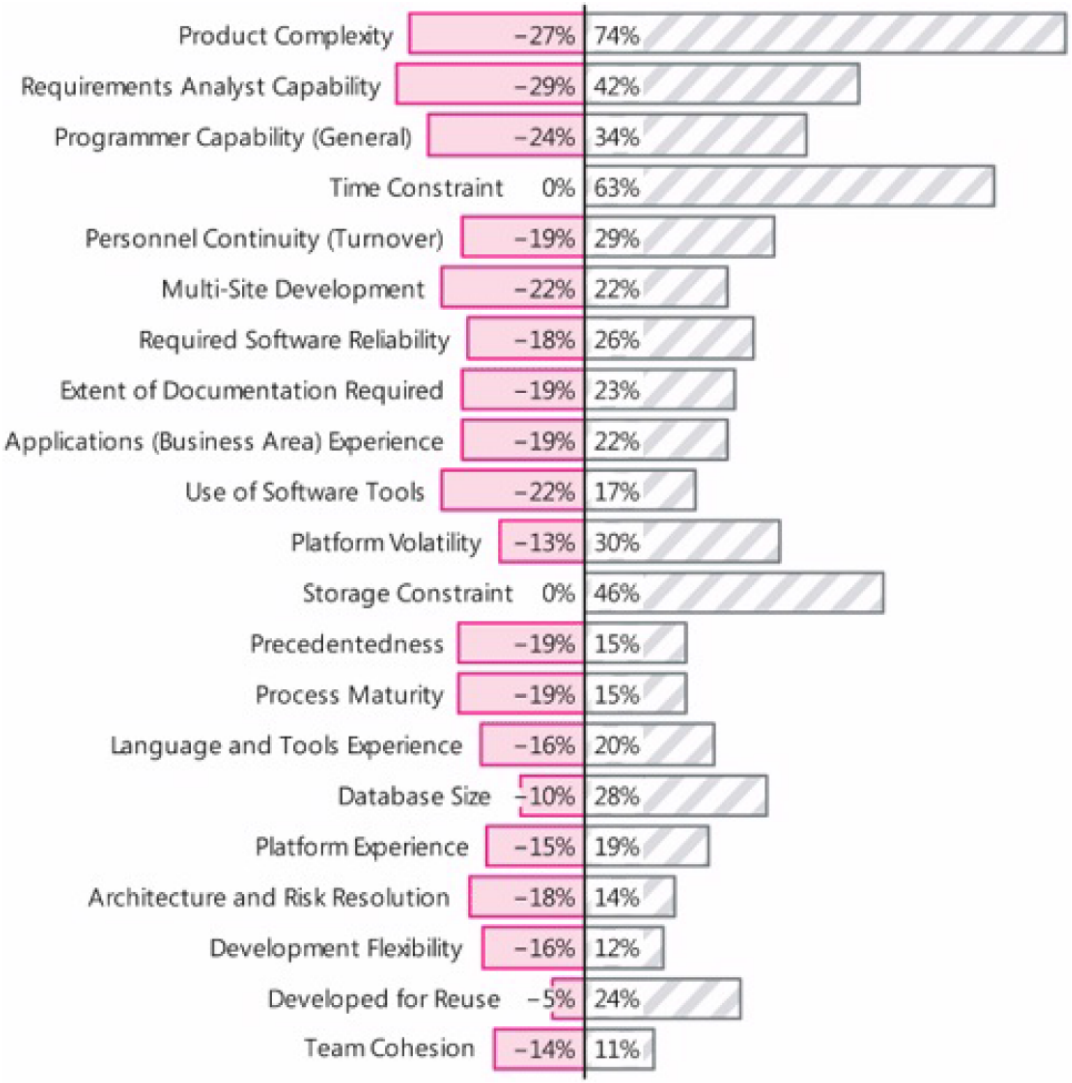
- Business applications, embedded systems, real-time

Personnel Factors

- Productivity of individuals can vary by a factor of 10 or more
- Mostly accounted for if estimates are based on team's historical performance

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Other Influences on Estimates



Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Data to Collect

Whenever possible count or compute your estimate rather than using expert judgment.

In order to compute software estimates based on historical data, we need to be collecting data about our projects.

- Project size
  - lines of code (LOC)
  - requirements
    - e.g., function points, scenarios, stories, UI screens
- Effort (staff months)
- Time (calendar months)

Collecting data on size and effort every 1 to 2 weeks can provide valuable insight into your project's dynamics.

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Calibration

Compute estimates of new development tasks based on the actual work to complete past tasks.

- Project data - data generated earlier in the same project that's being estimated
- Historical data - data from the organization that will conduct the project being estimated
- Industry data - data from other organizations that develop the same basic kind of software as the software that's being estimated

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimation by Analogy

Compute a size estimate based on a piece-by-piece count of analogous elements, and the past sizes of those elements.

Subsystem	Old Project (Size)	New Project (Est. Size)	Multiplier	Old Project (LOC)	New Project (LOC)
Use Cases	10 use cases	14 use cases	1.4	5,000	7000
User Interface	14 screens	16 screens	1.1	14,000	15,400
Graphs	10 graphs	16 graphs	1.6	9,000	25,600
Reports	3 reports	5 reports	1.7	3,000	5,100
Business Rules	???	???	1.5 (?)	11,000	16,500
TOTAL				**42,000**	**69,600**

Historical data
Estimates
Computed values

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Commercial Estimation Tools

Commercial tools embed computationally intensive estimation methods that cannot be done easily by hand or with a calculator.

- Simulating project outcomes (with respect to probability distributions for variability in system size, productivity)
- Probability analysis of estimates
- Complex effort estimations that account for diseconomies of scale

Tools tend to have large databases of industry-average data, but they can be calibrated with your own historical data.

Construx Estimate: www.construx.com/Resources/Construx_Estimate

Function Point Analysis

Goal: To estimate cost based on what we know at requirements time

We break the problem down into three parts:

1. Estimate the number function points from the requirements
2. Estimate code size from function points
3. Estimate resources required (time, personnel, money) from code size

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Function Points

A function point is a size metric of functionality. It relates to the requirements of the software under development.

The number of function points in a system is based on the number and complexity of the following

- **External Inputs** – screens, forms, dialog boxes, messages
- **External Outputs** – screens, reports, graphs, messages
- **External Queries** – responses to input requests that do not change system data
- **Internal Logical Files** – major internal data stores (end-user data, control information)
- **External Interfaces / APIs** – files or APIs controlled by adjacent systems

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Function Point Counting

A function point count is computed by adding up inputs and outputs, weighted by complexity multipliers.

Program Characteristic	Low Complexity	Medium Complexity	High Complexity
External Inputs	___ x 3	___ x 4	___ x 6
External Outputs	___ x 4	___ x 5	___ x 7
External Queries	___ x 3	___ x 4	___ x 6
Internal Files	___ x 4	___ x 10	___ x 15
External Interfaces / APIs	___ x 5	___ x 7	___ x 10

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimating Code Size from FPs

Language	Average	Median	Low	High
ASP	51	54	15	69
C	97	99	39	333
C++	50	53	25	80
C#	54	59	29	70
Excel	209	191	131	315
HTML	34	40	14	48
J2EE	46	49	15	67
Java	53	53	14	134
JavaScript	47	53	31	63
.NET	57	60	53	60
Oracle	37	40	17	60
SAS	38	37	22	55
SQL	21	21	13	37

http://www.qsm.com/resources/function-point-languages-table

Structured Expert Judgment

Use expert judgment as a last resort, or for a second or third estimation.

- People doing the work will create the most accurate estimates
- It's best to decompose estimation – and estimate tasks that require no more than 2 days of effort
- It's best to estimate a range (best case vs. worst case) of sizes
  - Single-point estimates tend to be much closer to best-case estimates
- Program Evaluation and Review Technique (PERT) Formula

ExpectedCase = [BestCase + (4*MostLikelyCase) + WorstCase] /6

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Summary of Size Estimation Tech.

Technique	Kinds of Sizes that can be Estimated
Analogy	features, function points, screens, GUI components, LOC
Estimation Tools	function points, LOC
Function Point Analysis	function points, LOC
Structured Judgment	features, user stories, requirements, use cases, function points, Web pages, GUI components, database tables, interface definitions, classes, functions/subroutines, LOC
GUI Elements	function points, LOC
Standard Components	function points, LOC
Story Points	story points, LOC
Wideband Delphi	features, user stories, requirements, use cases, function points, Web pages, GUI components, database tables, interface definitions, classes, functions/subroutines, LOC

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimating Effort

Informal Comparison to Past Projects – can compute an early estimate of effort (range of values) based on the effort results from analogous past projects

- Effort = PastEffort x (Size / PastSize)
- Linear model is OK, if relevant historical data is within a narrow size range

Project	Size (LOC)	Schedule (Calendar Months)	Effort (Staff Months)	Productivity (LOC/Staff Month)
Project A	33,842	8.2	21	1,612
Project B	97,614	12.5	99	986
Project C	7,444	4.7	2	3,722
Project D	54,322	11.3	40	1,358
Project E	340,343	24.0	533	639

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimating Effort

Formulae: If you don't have your own historical data, you can compute a rough estimate of effort based on models of industry-average efforts.

International Software Benchmarking Standards Group (ISBSG) Method

Desktop Projects:

$$\text{StaffMonths} = 0.157 \times \text{FunctionPoints}^{0.591} \times \text{MaxTeamSize}^{0.810}$$

Enhancement:

$$\text{StaffMonths} = 0.669 \times \text{FunctionPoints}^{0.338} \times \text{MaxTeamSize}^{0.758}$$

New Development:

$$\text{StaffMonths} = 0.520 \times \text{FunctionPoints}^{0.385} \times \text{MaxTeamSize}^{0.866}$$

Third Generation Language Project:

$$\text{StaffMonths} = 0.425 \times \text{FunctionPoints}^{0.488} \times \text{MaxTeamSize}^{0.697}$$

Fourth Generation Language Project:

$$\text{StaffMonths} = 0.317 \times \text{FunctionPoints}^{0.472} \times \text{MaxTeamSize}^{0.784}$$

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimating Schedule

There is a Basic Schedule Equation that produces good-enough estimates of schedule early in a medium-to-large sized project.

$$\text{ScheduleInMonths} = 3.0 \times \text{StaffMonths}^{1/3}$$

(where multiplier ranges from 2.0 to 4.0).

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimating Schedule

If you have historical data, you can compute a schedule estimate based on the schedules from similar past projects

Medium-to-large projects (more than 50 staff months)

$$\text{ScheduleInMonths} = \text{PastSchedule} \times (\text{EstimatedEffort} / \text{PastEffort})^{1/3}$$

Small projects

$$\text{ScheduleInMonths} = \text{PastSchedule} \times (\text{EstimatedEffort} / \text{PastEffort})^{1/2}$$

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

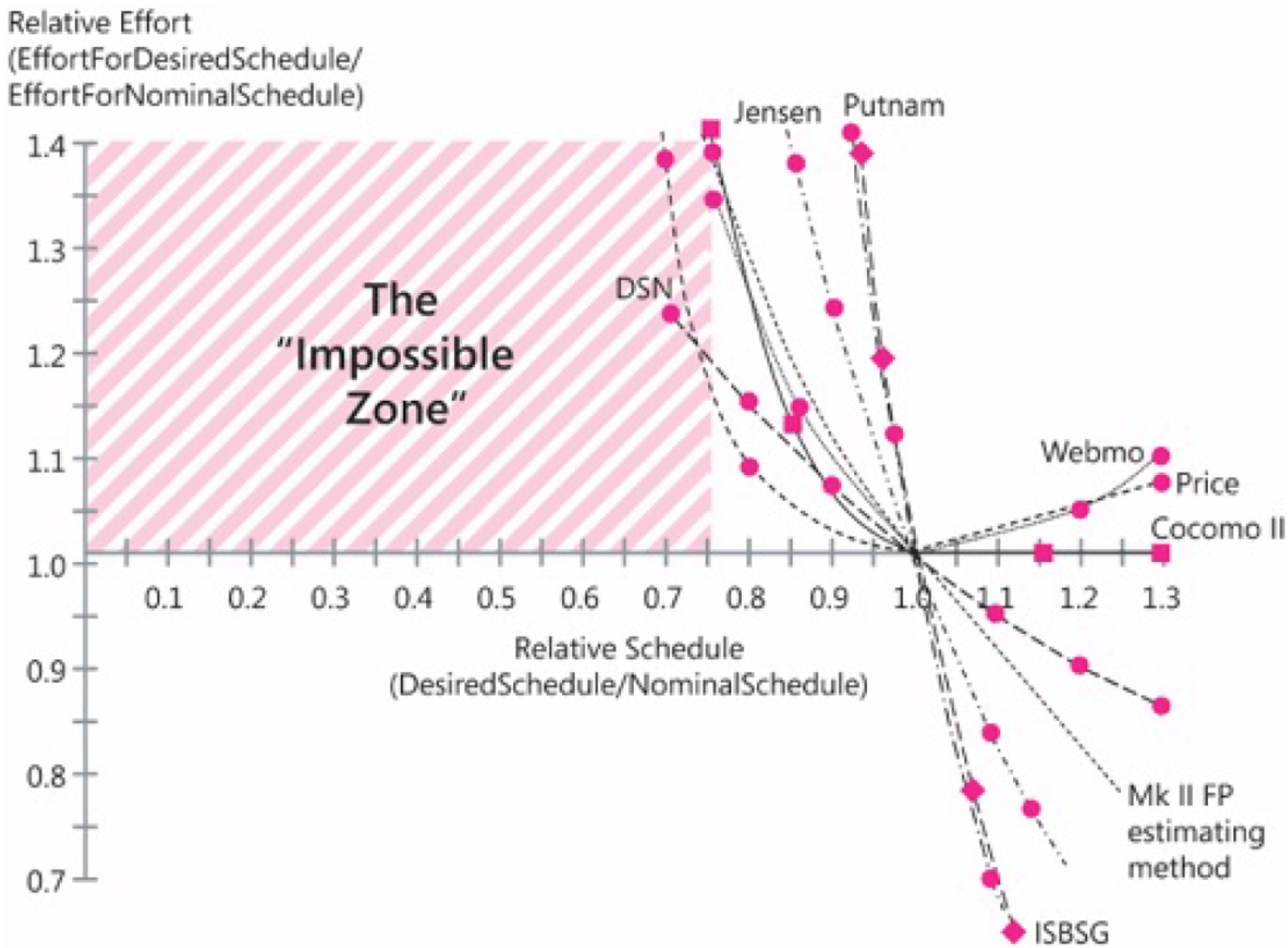
Estimating Schedule

Project	Size (LOC)	Schedule (Calendar Months)	Effort (Staff Months)	Productivity (LOC/Staff Month)
Project A	33,842	8.2	21	1,612
Project B	97,614	12.5	99	986
Project C	7,444	4.7	2	3,722
Project D	54,322	11.3	40	1,358
Project E	340,343	24.0	533	639

Suppose our estimated effort is 40 to 101 staff months, with an expected effort of 65 staff months.

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Schedule Compression



Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Schedule and Staffing Constraints

Schedule estimation techniques assume that, whatever the nominal schedule is, you'll be able to adjust your team size to fit the level indicated.

If staffing is fixed, then the only flexibility that you have is to reduce the feature set.

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

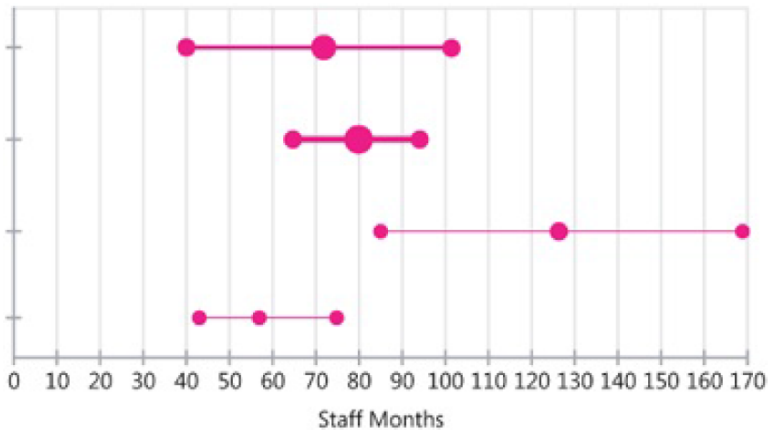
Best Practices

Decompose the estimation task into multiple subtasks and combine the results

- some estimation errors will cancel each other out

Use multiple estimation techniques and compare the results

- convergence suggests that you have good accuracy
- spread suggests some factors have been overlooked



Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Best Practices

Decompose the estimation task into multiple subtasks and combine the results

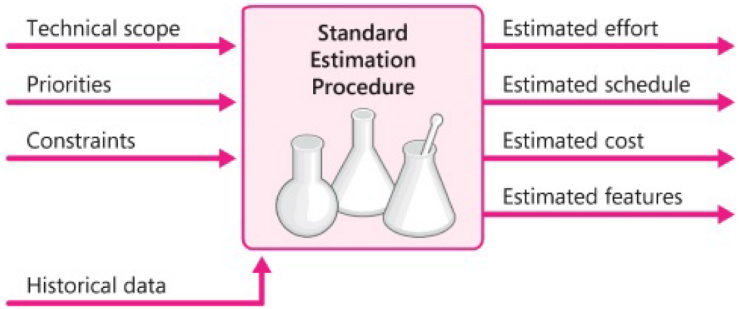
- some estimation errors will cancel each other out

Use multiple estimation techniques and compare the results

- convergence suggests that you have good accuracy
- spread suggests some factors have been overlooked

Define the estimation procedure in advance

- what granularity of task to count
- what multiple techniques to use
- when to re-estimate
- what project data to collect



Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimates as Probability Statements

Suppose you have a software estimate for a new project or iteration

Use Case	Best Case (staff months)	Expected Case (staff months)	Worst Case (staff months)
UC 1	4.1	5.0	6.0
UC 2	5.2	6.4	7.7
UC 3	6.0	7.1	8.2
UC 4	2.4	3.1	3.6
UC 5	5.8	7.0	8.2
TOTAL	**23.5**	**28.6**	**33.7**

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimates as Probability Statements

Goal:

Percentage Confident	Effort Estimate (staff months)
2	23.5
10	25.4
16	26.1
20	26.5
25	26.9
30	27.3
40	28.0
50	28.6
60	29.3
70	30.0
75	30.3
80	30.8
84	31.2
90	31.8
98	33.7

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimates as Probability Statements

1. Combine the standard deviations of the piecemeal estimates by summing the variances of the estimates and taking the square root.

Use Case	Best Case (BC)	Worst Case (WC)	Standard Deviation (WC-BC)/6	Variance (SD^2)
UC 1	4.1	6.0	0.317	0.100
UC 2	5.2	7.7	0.417	0.174
UC 3	6.0	8.2	0.367	0.135
UC 4	2.4	3.6	0.20	0.040
UC 5	5.8	8.2	0.40	0.160
TOTAL	**23.5**	**33.7**		**0.608**
Combined Standard Deviation				**0.78**

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimates as Probability Statements

2. Can use a table of standard deviations to compute a percentage likelihood.

Percentage Confident	Calculation	Percentage Confident	Calculation
2	Expected Case – (2 x SD)	60	Expected Case + (0.25 x SD)
10	Expected Case – (1.28 x SD)	70	Expected Case + (0.52 x SD)
16	Expected Case – (1 x SD)	75	Expected Case + (0.67 x SD)
20	Expected Case – (0.84 x SD)	80	Expected Case + (0.84 x SD)
25	Expected Case – (0.67 x SD)	84	Expected Case + (1 x SD)
30	Expected Case – (0.52 x SD)	90	Expected Case + (1.28 x SD)
40	Expected Case – (0.25 x SD)	98	Expected Case + (2 x SD)
50	Expected Case		

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimates as Probability Statements

The above assumes that your piecemeal estimates have a confidence interval of 99.7% (which might be optimistic!)

More realistic estimates of the SDs of the piecemeal estimates will consider the best-case to worst-case range to have a smaller confidence interval (and thus a SD is a larger fraction of the range)

Confidence Interval	Divisor in calculations of SDs of piece estimates
10%	0.25
20%	0.51
30%	0.77
40%	1.0
50%	1.4
60%	1.7
70%	2.1
80%	2.6
90%	3.3
99.7%	6.0

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimates as Probability Statements

Use Case	Best Case (BC)	Worst Case (WC)	Standard Deviation (WC-BC)/2	Variance (SD^2)
UC 1	4.1	6.0	0.95	0.902
UC 2	5.2	7.7	1.25	1.56
UC 3	6.0	8.2	1.1	1.21
UC 4	2.4	3.6	0.6	0.36
UC 5	5.8	8.2	1.2	1.44
TOTAL	**23.5**	**33.7**		**5.48**
Combined Standard Deviation				**2.34**

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Estimates as Probability Statements

Percentage Confident	Effort Estimate (staff months)
25	28.6 – (0.67 x **2.34**) = ~~27.0322~~ 27
50	28.6
75	28.6 + (0.67 x **2.34**) = ~~30.1678~~ 30
90	28.6 + (1.28 x **2.34**) = ~~31.5952~~ 32

Lastly, try to report estimates in significant digits and units that are consistent with the estimate's underlying accuracy and precision

Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

Summary

Software estimation is critical in providing input to decisions about project scoping, planning, and tracking.

- Estimations vs. targets vs. commitments
- Techniques for estimating project size, effort, and schedule
- Sources of inaccuracies in estimations
- Data collection to support calibration in techniques
- Best practices
- Probability ranges and precision

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Software Estimation for FYDP

Data Collection

In order to compute software estimates based on historical data, you need to be collecting data about your project (every 1-2 weeks)

Per requirement (e.g., use case, scenario, story, screen)

- lines of code (LOC)
- effort (staff hours)
- time (calendar hours)

Software Estimation

- effort of whole project (as sum of efforts of use cases)
- effort of next increment (as sum of efforts of atomic requirements)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Deliverable #11

Requirements-Based Estimate of Effort

Estimate the effort to develop 2 top-priority use cases, based on expert-judgment estimates of the effort to develop details about the use cases

- atomic specifications (conditions of satisfaction)
- associated UI screens

Scenario-Based Estimate of Effort

Estimate the effort to develop the rest of the use cases in your project, based on function point analysis of those use cases

Percentage Confidence in Estimates

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Requirements & Specs at Roadmunk

JULY 14, 2016

COLIN RHODES

roadmunk™

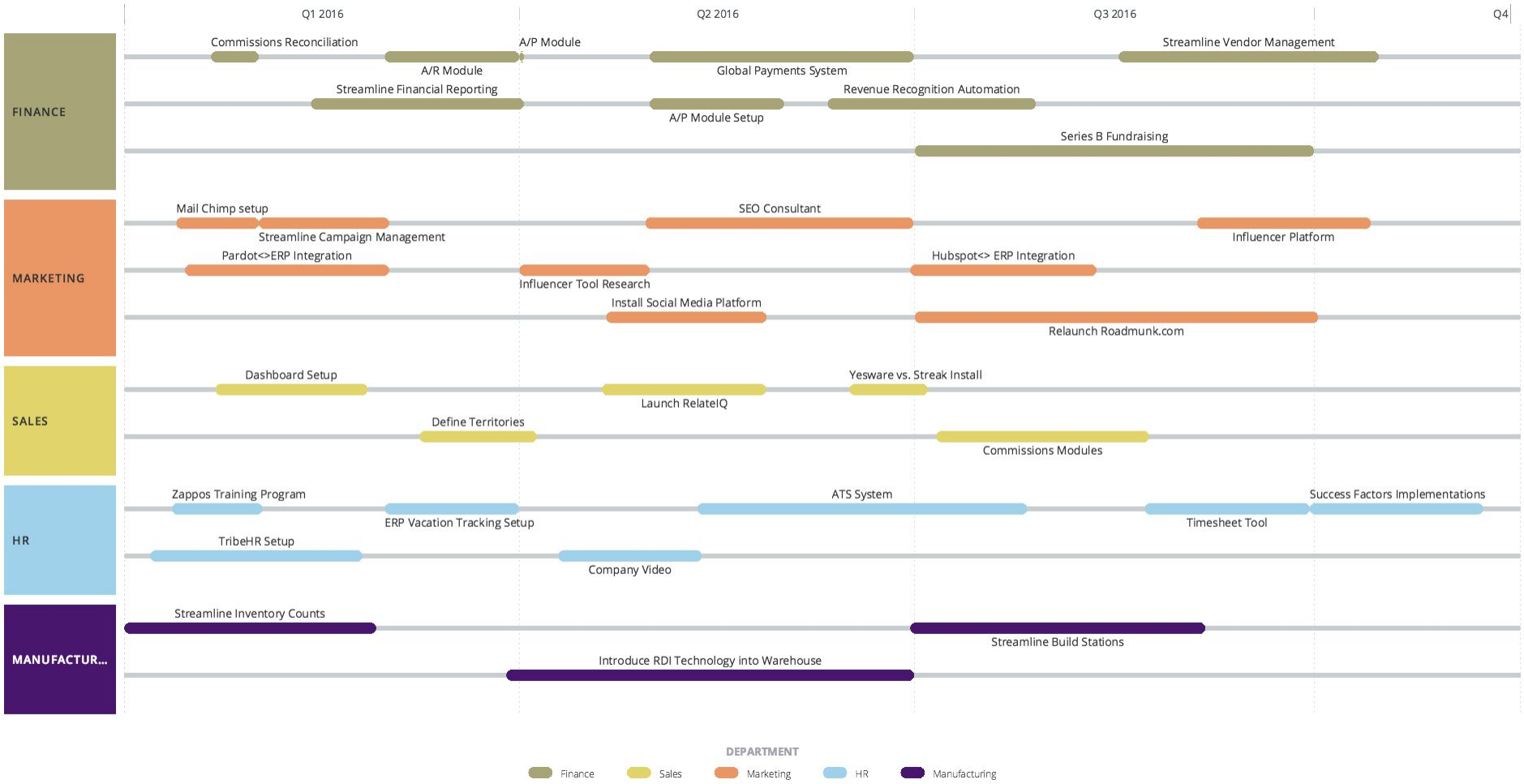
Colin Rhodes

- SoftEng 2008
- Miovision 2008-2014, Kitchener
  - traffic engineering systems
  - automated data collection
  - software + hardware
  - role: Software Developer
  - started when 6 ppl, left when 90 ppl
- Roadmunk 2014-date, Kitchener
  - product roadmaps, SaaS
  - role: Software Developer
  - started when 3 ppl, now 17

About Roadmunk

- Web app for Product Managers to create beautiful roadmaps quickly
- Multiple visualizations and pivots on the same source data
- Sharing & collaboration
- Competitive advantage:
  - slick, uncomplicated UX
  - highly responsive support
- remote culture
- Costa Rica trip

Sample Roadmap View



roadmunk™

Sample Roadmap View

	MOBILE	WEB APPLICATION	DESIGN / UX	ANALYTICS
ALPHA RELEASE	Drag and Drop Tools Google Single Sign Slider Menu	Integration with Roadmunk Stripe for Mobile Payments	Upgrade Icons to IconMoon Flat UI Custom Color Palettes	CRM Reporting Analytics Engine
BETA RELEASE	Testing System Cross Browser Testing Suite	SEO Moz Buffer Integration	Design Admin Backend	SQL DB Streamline Google Analytics Integration
GA RELEASE	iPhone app (MVP) Responsive Homepage	Website re-tooling Filtering on Directory Page	Profiling of team Squarespace Setup for Blog Homepage Redesign on Wordpress Product Video with Feature Overview	Operations Setup Build Project Management tool Setup ZenDesk
FUTURE	Stabilization on List Preview and Searching	Update App to New Navigation Menu		Year-end Support for Finance

RELEASE

Future | GA Release | Alpha Release | Beta Release

Published on **November 5 2015**

Designed with **Roadmunk**

roadmunk™

TEAM STRUCTURE

- Split Office
  - Toronto: sales, marketing, product, design
  - Kitchener: dev (6), QA (1)
- Co-founders are mid-30s with involvement in 2-3 previous startups each
- Product Manager
  - what should we build?
  - high-level workflow
- Designer
  - how should it look?
  - detailed workflow

Roadmunk Process

1. Backlog Management
2. Feature Ideation
   - output: rough spec
3. Functional Prototype
   - output: working prototype
4. Detailed Design
   - intermediary output: pretty mocks
   - final output: finished feature

1. Backlog Management

roadmunk™

Tracking Ideas

"What problem are you trying to solve?"

1. Record ideas for projects/features

2. Determine market value
   - total requests
   - enterprise deal blockers
   - churn reduction

3. Prioritize and investigate further

Backlog Sources

- Tag every support case
- Customer calls & face-to-face meetings
- Competitive analysis
- Big spreadsheet to track number of requests for each feature by month

Sub-roadmapping	123
JIRA Sync Enhancements	214
More Integrations	122
Dependency Tracking	100
Existing Functionality	766

2. Feature Ideation

roadmunk™

Feature Idea Tracking

- Brainstorm ways to solve problem
  - team: product manager, senior devs
- Define use cases and prioritize
  - required so that when we debate UX, we can back up our arguments
- Scoping: MSP vs MVP
- Iterate towards written overview of workflow
- Rough wireframes
- Interactive mocks as needed
- Buy-in from internal stakeholders
- Validation with customers

Sample Use Cases

P1

- As an editor, I want to **create shared fields** that may be accessed by all users on my account, so that we can centrally manage shared fields in the org
- As an editor, I want to **merge** a **shared field** with a non-shared field, so that I can decrease the duplication of the same fields and centrally manage the fields

P2

- As an **editor**, I'd like to be able to **reuse fields** across my different roadmaps but not necessarily expose them to the entire account

MSP vs MVP

MSP: Minimum Sellable Product

- enough features to convince someone to buy

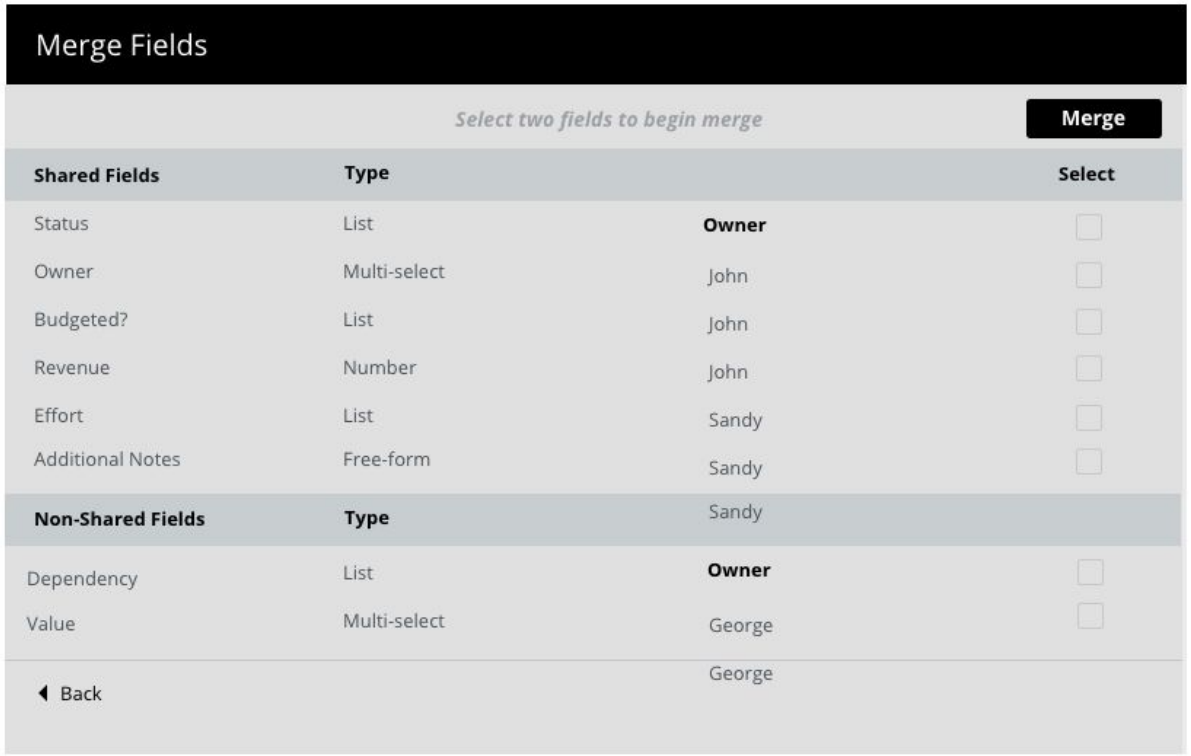
MVP: Minimum Viable Product

- enough features for most users to be satisfied when they actually use the product

Feature sets overlap, but often don't have subset relationship.

Rough Wireframes

- Avoid spending too much time on details
- Re-use existing widgets
- Goal is to define workflow rather than aesthetics



Interactive Mocks

Clickable mockups that demonstrate workflow & navigation

- More work to create than static
- Allow designer to identify more problems before presenting to devs
- Great for validation with customers

Tools: Framerjs, Pixate, ...

3. Functional Prototype

roadmunk™

Why Functional Prototype?

- Designer was wasting time defining details before the workflow validated
- Devs were wasting time building details before workflow validated
- Hard to catch all workflow issues without having something to play with

Efficient Prototyping

- Focus on workflow, not visuals
- Re-usable widgets!
- Make UI clean enough to not distract from the workflow
- Iterate as necessary
  - build prototype
  - automated & manual testing
  - review with sr dev, product, design
  - revise

4. Detailed Design

roadmunk™

Detailed Design

- Start after workflow is finalized
- Designer can make pretty mocks
- Be prepared to back up all decisions against our design philosophy
- Implement
- Tweak with dev & designer at one computer
- Repeat manual testing
- Targeted beta vs full release?

Variants

roadmunk™

How Miovision was Different

- Multiple dev teams coordinating on projects
- Several systems coordinating
- Longer release cycles
- More focus on data & reporting than UX

Result:

- No prototyping phase
- More work into up-front designs with detailed mocks
- Clear contracts between systems

Review

- Backlog Management
- Feature Ideation
- Functional Prototype
- Detailed Design

Retrospectives & Iterate on Process!

Questions?

roadmunk™

SE463
Software Requirements Specification & Analysis

RE Reference Model

Readings:

Jackson, M. and Zave, P. "Deriving specifications from requirements: an example". in *Proceedings of the 17th international Conference on Software Engineering (ICSE),* 1995, pp. 15-24.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Objectives

Want to identify and articulate

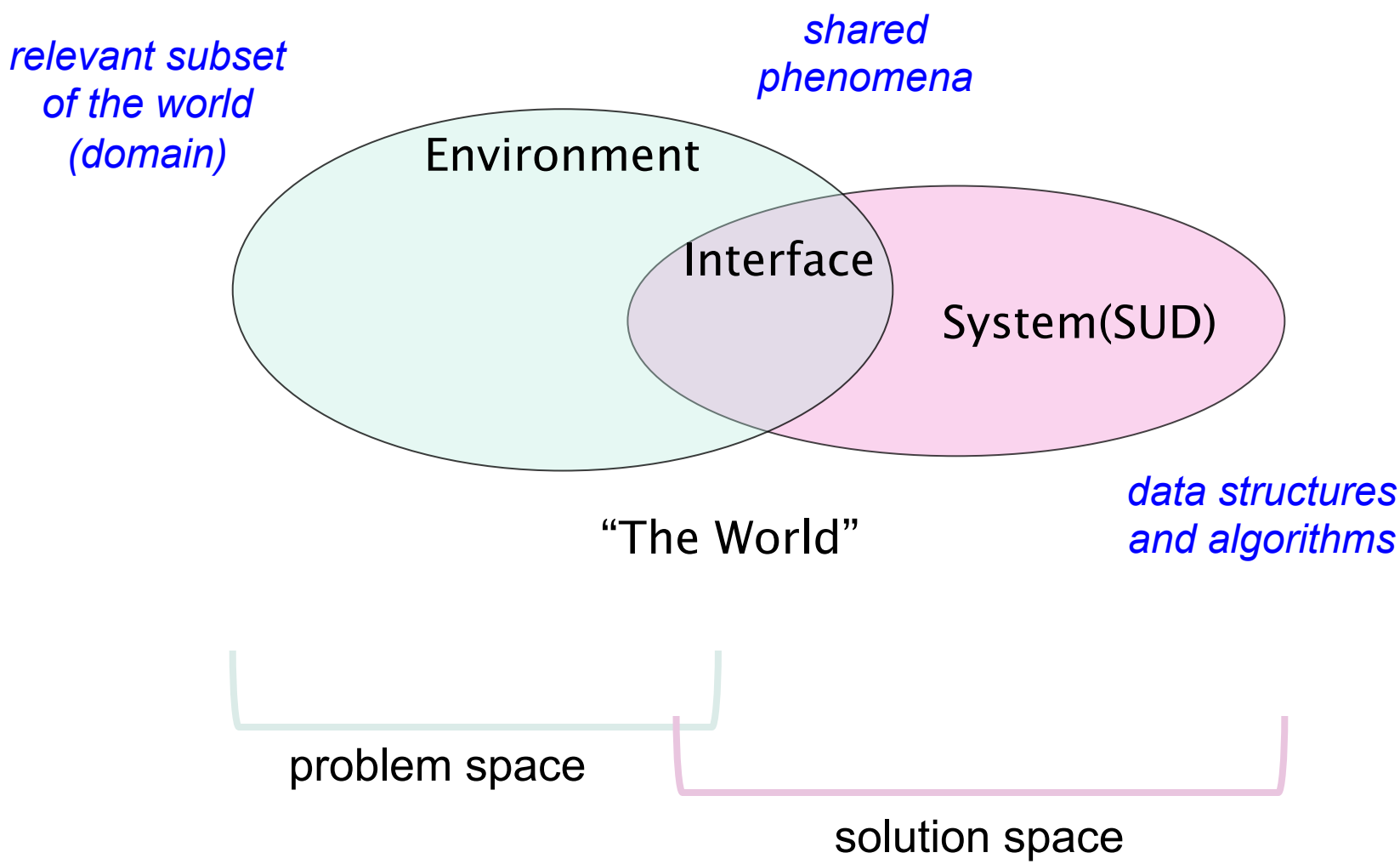
- Requirements - Conditions and capabilities that describe a *problem* – to be met by a *solution*, for the solution to be acceptable

- Specification - A complete, precise, verifiable expression of requirements of a *software* or *system* solution.

in a way that avoids implementation bias.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Problems vs. Solutions

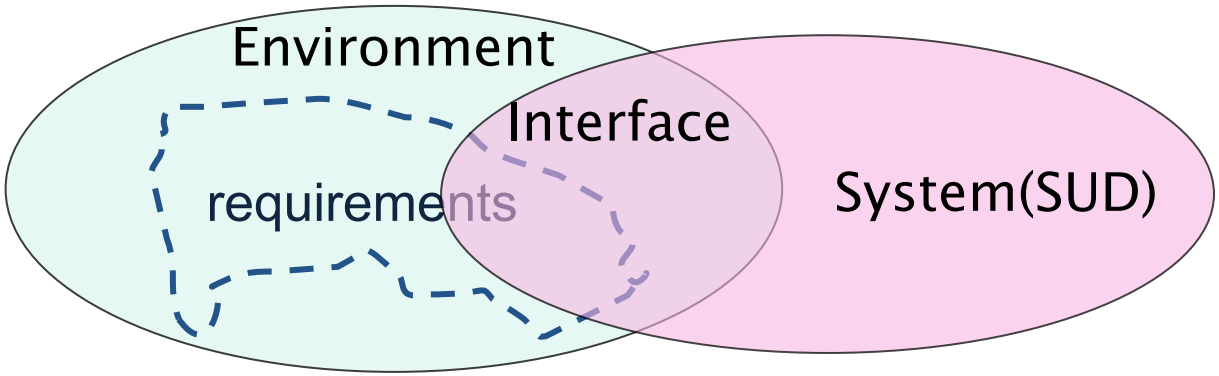


Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Requirements

A requirement is a condition or capability that must be achieved

- desired changes to the World
- expressed in terms of environmental phenomena



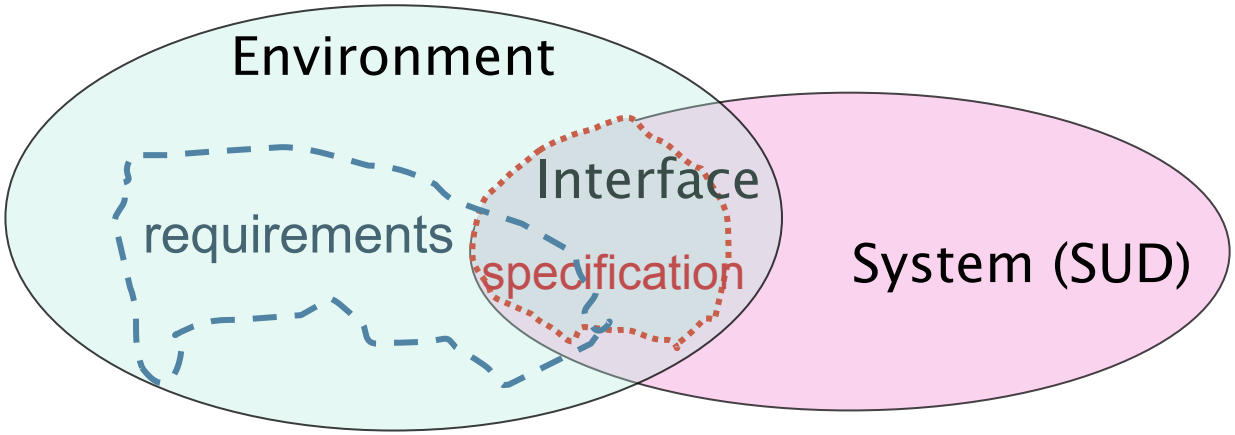
"The World"

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Specification

A specification is a description of the proposed system

- desired changes to the world
- requirements re-expressed in terms of interface phenomena
- places no constraints on the design or implementation of the system giving the designer maximum freedom



"The World"

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Park User Fees

Suppose that the city of Waterloo decides to raise funds by instituting users fees for public parks.

Requirements:

R1: Collect $1 fee from each user on entry to the park.

R2: Ensure that anyone who has paid may enter the park.

R3: Ensure that no one may enter park without paying.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Domain Knowledge

Ideally, we want to be able to show that the specifications imply the requirements:

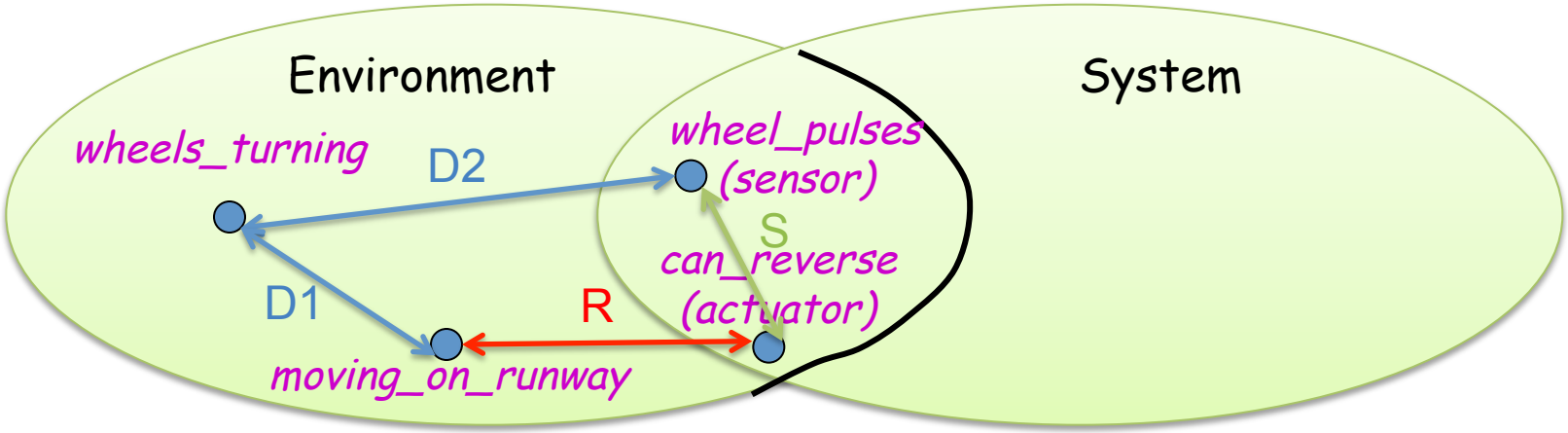
$$\text{Spec} \models \text{Req}$$

Often we cannot do so without making some assumptions about how the environment behaves.

$$\text{Dom} \subseteq \text{Env}$$

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Real World Example

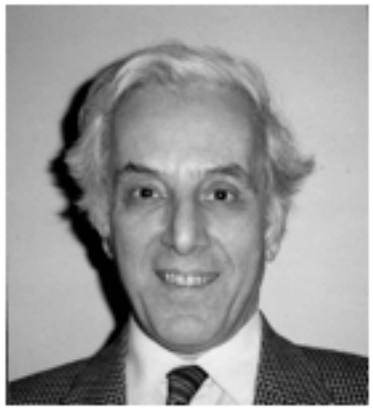


R: An airplane may engage reverse thrust iff it's moving on the runway

D1: Moving on runway iff wheels turning

D2: Wheel pulses detected iff wheels turning

S: Can reverse iff wheel pulses detected



Michael Jackson, *Software Requirements and Specification*, 1995

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Park User Fees Example

Requirement	Interface	Specification
Collect $1 fee from each user on entry to the park.	Coin slot	(Env) coin inserted into slot
		(Sys) senses coin
Ensure that anyone who has paid may enter the park	Barrier	(Sys) unlocks barrier upon sensing a new coin
		(Env) visitor pushes unlocked barrier
Ensure that no one may enter park without paying	Barrier	(Sys) detects entry
		(Sys) relocks barrier

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

RE Reference Model

The fundamental law of requirements:

$$\text{Dom}, \text{Spec} \models \text{Req}$$

Must be able to argue that the specification of the system plus the assumptions are enough to satisfy the requirements.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Deriving Specifications

For each requirement Req

- Determine how system will monitor / control the environment
  - may need to introduce interface (i.e., sensors, actuators)
  - recast Req in terms of interface phenomena (the result is a Spec)
- Determine whether Req constrains environmentally-controlled phenomena
  - if so, identify domain assumptions (Dom) that link the environmentally-controlled phenomena to system-controlled phenomena
- Check that $\text{Dom}, \text{Spec} \models \text{Req}$

  $\text{Dom} \wedge \text{Spec}$ is satisfiable

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example

Example #1: Thermostat

R: Want to keep the air temperature at or above the set temperature.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example

Example #2: Traffic Light

R: Allow car traffic to cross an intersection safely, without colliding with traffic travelling in other directions.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example

Example #3: Patient Monitor

R1: Retain records of the patient's vital signs

R2: Warn intensive-care nurse if the patient's readings exceed the safe ranges.

[Stevens, Myers, and Constantine, "Structured Design", *IBM Systems Journal*, 13(2), 1974.]

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Reference model for requirements engineering

- Terminology
- Deriving specifications from requirements
- Spec, Dom $\models$ Req

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Scope, Stakeholders, and Goals

Readings:

Robertson, S. and Robertson, J., *Mastering the Requirements Process, 3rd ed.,* Chapter 3 "Scoping the Business Problem"

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Project Types

smaller projects, short lifetimes, small number of stakeholders, iterative development with customer in close proximity

projects of medium longevity, dozen+ stakeholders, geographically distributed

large projects, outsourced development, or need for certification

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

The Problem Is To Know What the Problem Is

"If I had only one hour to save the world, I would spend fifty-five minutes defining the problem, and only five minutes finding the solution."

- Albert Einstein

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Scoping the Problem

Purpose of the project: a rationale for *why* the project is wanted, what benefit it brings to the business.

Goal: Some high-level measureable criterion of success

The scope of the work: The business area affected by the installation of the system. You need to understand this work to specify the most appropriate product.

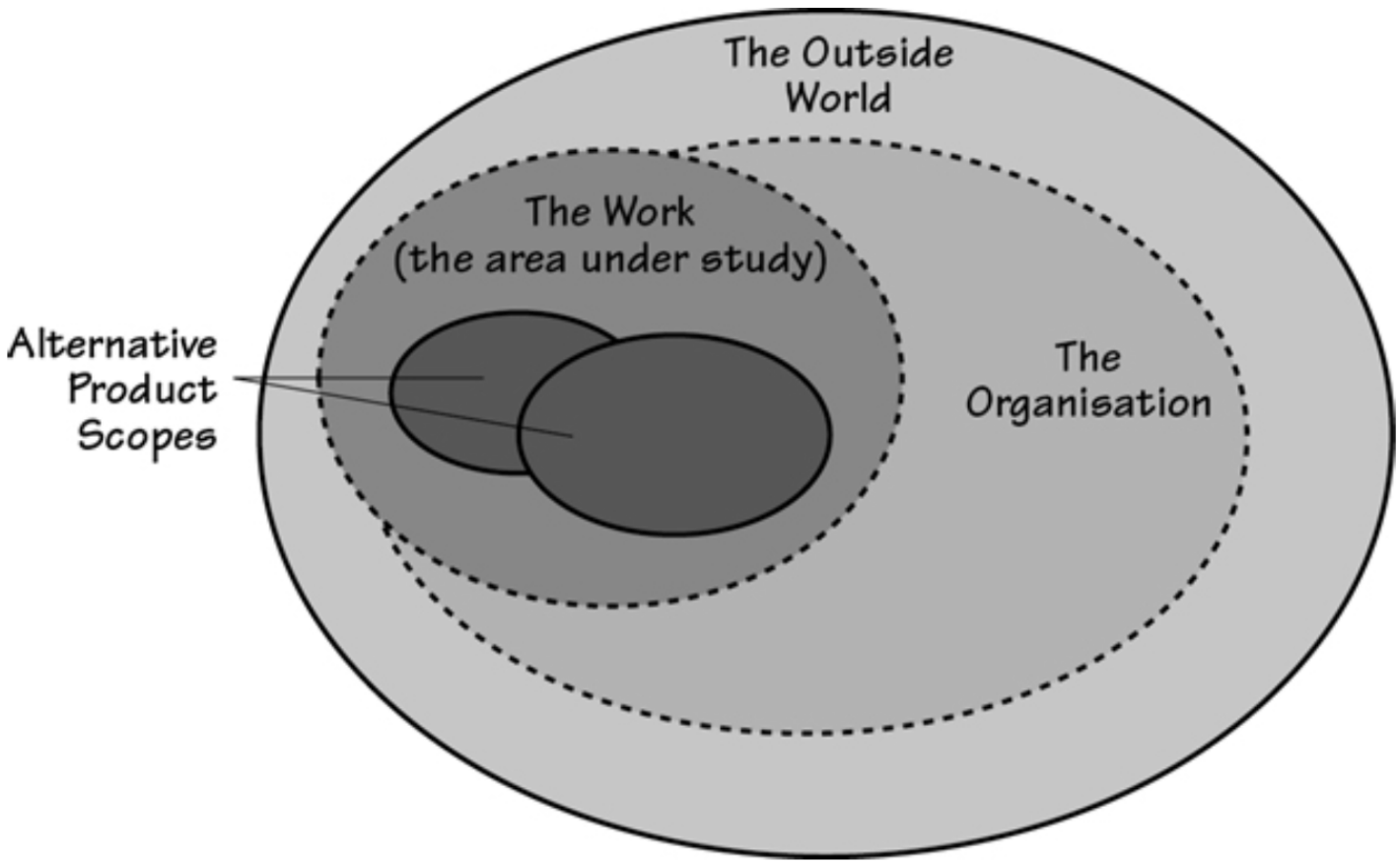
The stakeholders: The people with an interest in the system.

Constraints: Restrictions on the scope or style of the system.

- predetermined design solutions that must be used
- constraints on changing current business processes
- time and money that are available for the project

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

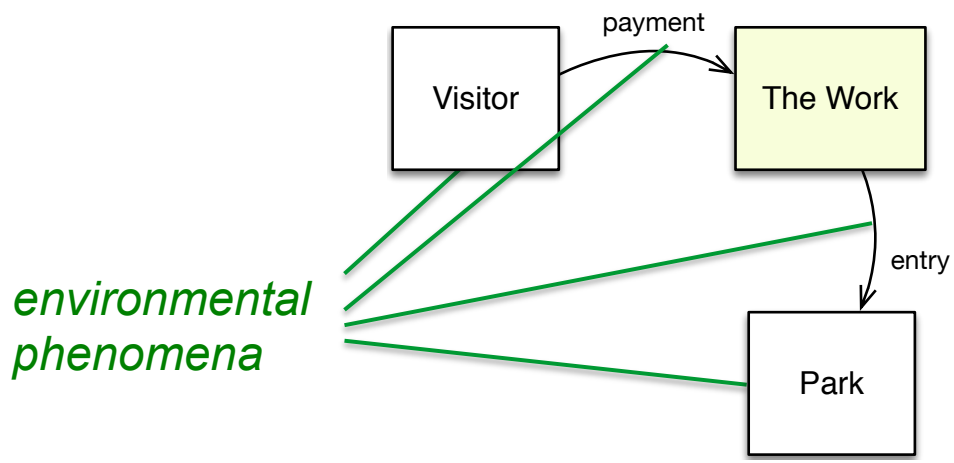
Scoping the Problem



Robertson, Robertson, *Mastering the Requirements Process*, 2012, Figure 3.3.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Context Diagram

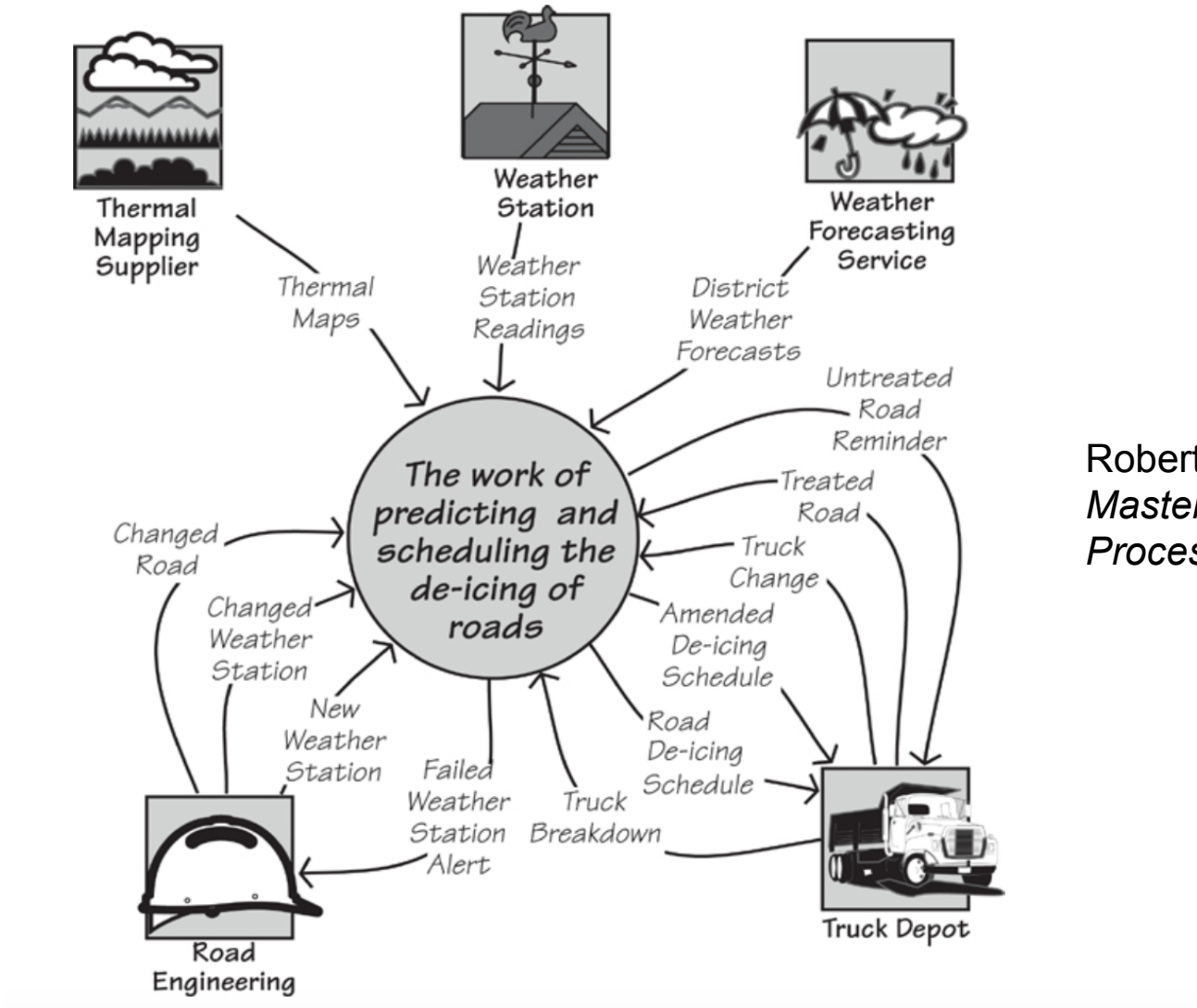


A context diagram is a graphical model of the context (Environment) in which the Work exists

- Phenomena that are relevant to the descriptions of the requirements
- Modularizes the phenomena into domains.
- Precursor to Domain Models (next week)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Context Diagram



Robertson, Robertson, *Mastering the Requirements Process*, 2012, Figure 3.5.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Context Diagram

Patients in an intensive-care ward in a hospital are monitored by sensors of various kinds (i.e., by electronic analog devices attached to their bodies). Through the sensors, the devices measure the patient's vital factors: pulse rate, temperature, blood pressure, and so on. A program is needed to read the factors, at a frequency specified for each patient, and store them in a database. The factors read are to be compared with safe ranges specified for each patient, and readings that exceed the safe ranges are to be reported by alarm messages displayed on the screen of the nurse's station.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Early Deliverables

Project Purpose

Not a description of *what* the proposed project will do, but a rationale for *why* it is wanted

Measureable Goal

Some high-level *measureable criterion* for what it means for your project to be deemed a success (i.e., fit for purpose)

Technical Feasibility

(1) Evidence of a promising solution that meets the measurable goal and (2) the organization (team) has or can acquire the skills to build the system

Constraints

Restrictions on acceptable solutions

- pre-ordained design decisions (platform, OS, existing database, COTS, etc.)
- project constraints, like limits on budget or development time

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: JobMine

Problem Statement:

- Facilitates the scheduling of job interviews.
- Matches job applicants with job openings, according to applicants' and employers' preferences.

Project Purpose

Measureable Goal

Technical Feasibility

Constraints

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

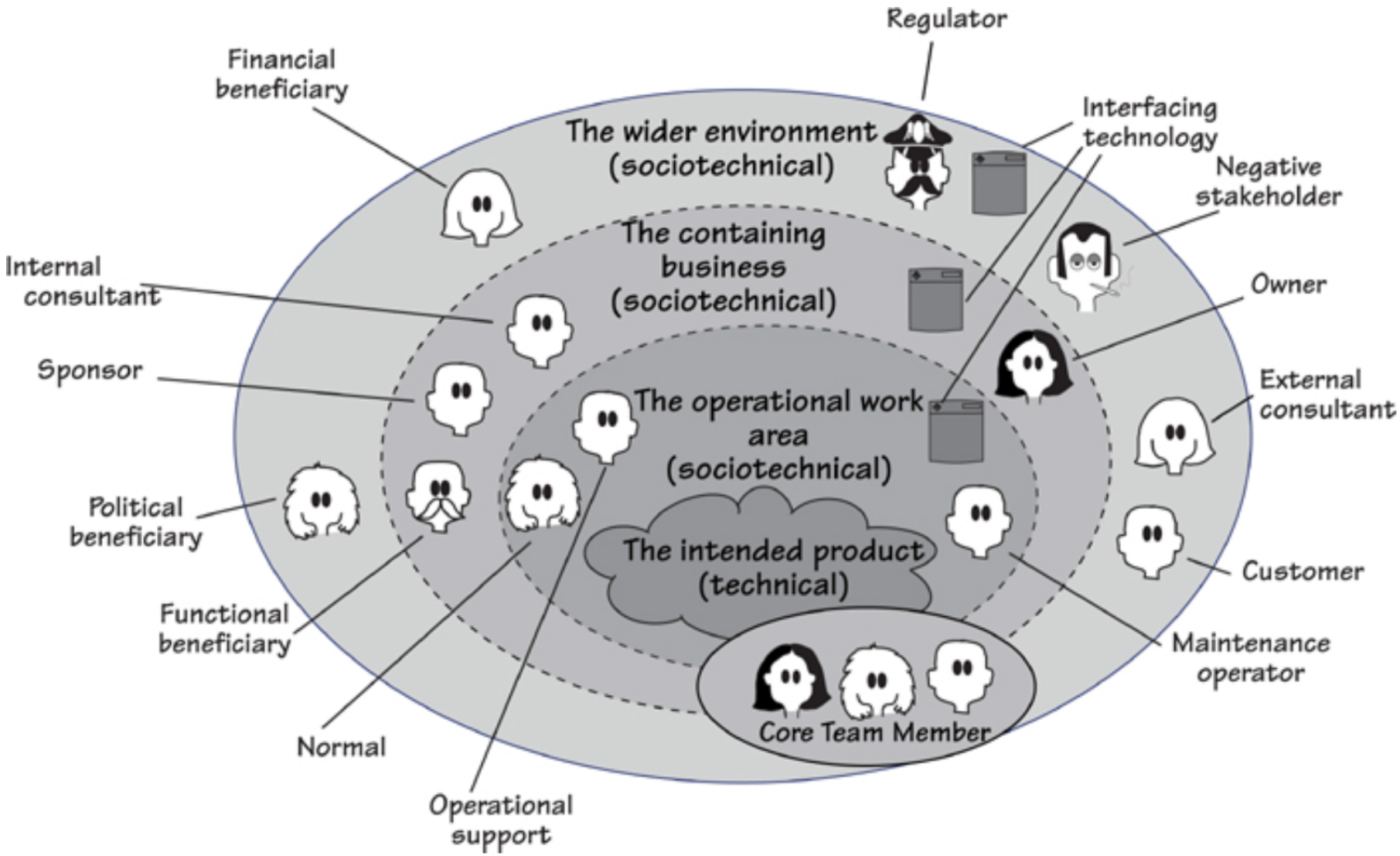
Stakeholders

- A stakeholder is anyone who has a stake in the ultimate success of the project.

- During elicitation, we want to talk to all of the stakeholders.
  - But sometimes we have to make do with proxies

© 2007 Oxford Creativity

Stakeholders



Robertson, Robertson, *Mastering the Requirements Process*, 2012, Figure 3.7.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Stakeholders: Owner/Client

- The *owner/client* is the person paying for the software to be developed
  - They are the ultimate stakeholder — almost always, the client has the last say in what the product does
  - For “bespoke” or customized systems, the client is the person with the chequebook ☺
  - For software for the mass market, the client may be the company developing the software
  - For in-house software, the client is probably the manager of the product’s users
    - Since his/her employees will be the primary beneficiaries, it is reasonable for him/her to pay for the project
  - Likely represented by a proxy

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Stakeholders: Customer

- A *customer* is a person who buys software after it is developed; possibly a manager
  - May be the same as the client
  - May be the same as the user; other times, the customer is an office manager who buys software for his / her staff
    - For what requirements will he / she pay? Which are trivial or are excessive?
  - Must be an active participant in the project (or have an active representative if there are many customers)
    - Sometimes a marketing person or experienced manager serves as a proxy

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Stakeholders: Users

- *Users* (of both the current and future systems)
  - Experts on the existing system — tell us which features to keep and which need improvements, or
  - Experts on competitors' products — give suggestions about how to build a superior product.
  - May have special needs or requirements to be satisfied, e.g., regarding usability, desired features.
  - May want to consult special-interest groups: users with disabilities, users who have computer phobias, expert users, novice users, older users, etc.
  - May be simulated by a persona.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Diversity Matters!

Different users make different demands on your project.

1) What categories of users do you need to consider?
   - e.g., normal users, operational support, maintenance

2) For each category of user, describe their background, skills, experience

3) For each category of user, identify the accessibility characteristics that your project must cater to

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Stakeholders: Domain expert

- *Domain experts* know the problem domain well
  - Familiar with the problem that the software must solve.

    e.g., financial experts for financial packages,
    aeronautical engineers for aircraft navigation systems,
    meteorologists for software that models the weather,
    travel agents for travel industry, etc.
  - … and are familiar with typical users and their expectations
  - … and are familiar with typical deployment environments

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Stakeholder: Software engineer

- *Software engineer* == technology expert
  - May include managers too
  - Represents the rest of the development team (developers, testers)
  - Ensures that the project is technically and economically feasible
  - Accurately estimates the cost and development time of the product
  - Educates the customer about innovative hardware or software technologies, and recommends new functionality that takes advantage of these technologies

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Other Stakeholders

- *Inspectors* == experts on government and safety regulations
  - Familiar with government and safety regulations relevant to project
    e.g., safety inspectors, auditors, certification inspectors
- *Market researchers*
  - Experts who have conducted surveys to determine future trends and potential customers' needs.
  - May assume the role of client, if the software is being developed for the mass market and there is no identifiable customer.
- *Lawyers*
  - Familiar with legal requirements
  - Familiar with licensing, e.g., for use of open source components
- *Industry standards*

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Other Stakeholders

- *Experts on adjacent systems*
  - They know about the interfaces to adjacent systems, and any special demands for interfacing with the adjacent system.
  - They need to be forewarned about the planned new system -- they could be affected by your work!
  - They can help to improve your work, by enhancing the services they provide.
- *Negative stakeholder*
  - It is best to understand why they do not want the project to succeed
  - They may simply have competing requirements that you'd be best off considering

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

© 1993 Groening. Dist. By Fox Network

Early Deliverables

Stakeholders

- Types, Names, What information they can provide
- Categories of users
- For each category of user, describe their background, skills, experience
- For each category of user, identify their accessibility characteristics

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Volere Req. Spec. Template

Project Drivers

1. The Purpose of the Project
2. The Stakeholders

Project Constraints

3. Mandated Constraints
4. Naming Conventions and Terminology
5. Relevant Facts and Assumptions

Robertson, Robertson, *Mastering the Requirements Process*, 2012, Appendix A.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Earliest stage of requirements engineering

- Scoping the problem
- Context diagram
- Identifying the stakeholders
- Volere Requirements Specification Template

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Business Use Cases

Readings:

Robertson, S. and Robertson, J., *Mastering the Requirements Process, 3ed.*, Chapter 4 "Business Use Cases"

AND

Larman, *Applying UML and Patterns*, Chapter 6: "Use Cases"

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Furniture Design Exercise

Working in groups of 3 and 4, design an innovative piece of furniture that can be manufactured by individuals who have physical or mental impairments, disabilities, or conditions.

Name Requirements:

It can serve as a chair, footstool, room divider, coffee table, bookshelf, and end table.

Functional Requirements:

It can be used to sit on, prop feet on, divide rooms, and put books, lamps, magazines, and knickknacks on.

Constraints:

cost (must sell for under $50)

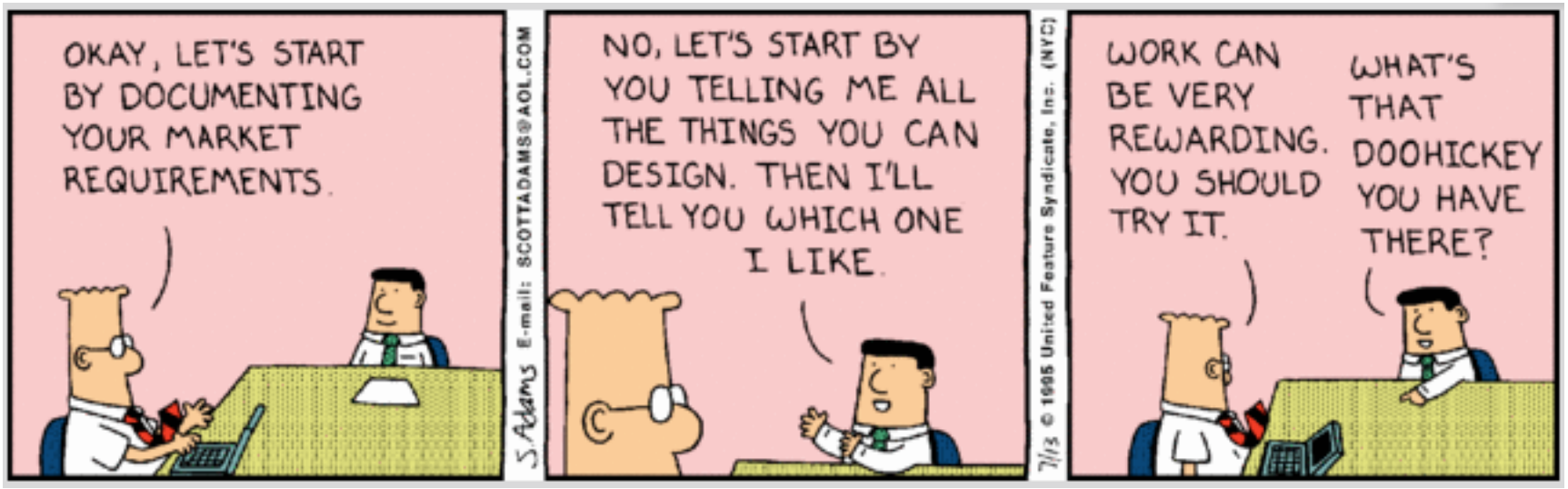
shipping (must ship unpackaged without fear of damage)

Evaluation Criteria:

innovative

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Business Requirements



© 1995 Scott Adams, Inc. / Dist. by United Feature Syndicate, Inc.

"If your software does not have to satisfy a need, then you can build anything.

However, if it is meant to satisfy a need, then you have to know what that need is to build the right software."

Robertson, Robertson, *Mastering the Requirements Process*, *3ed.,* 2012

Business Use Cases

Understanding the work – understand the work that is to be improved before offering a solution or product.

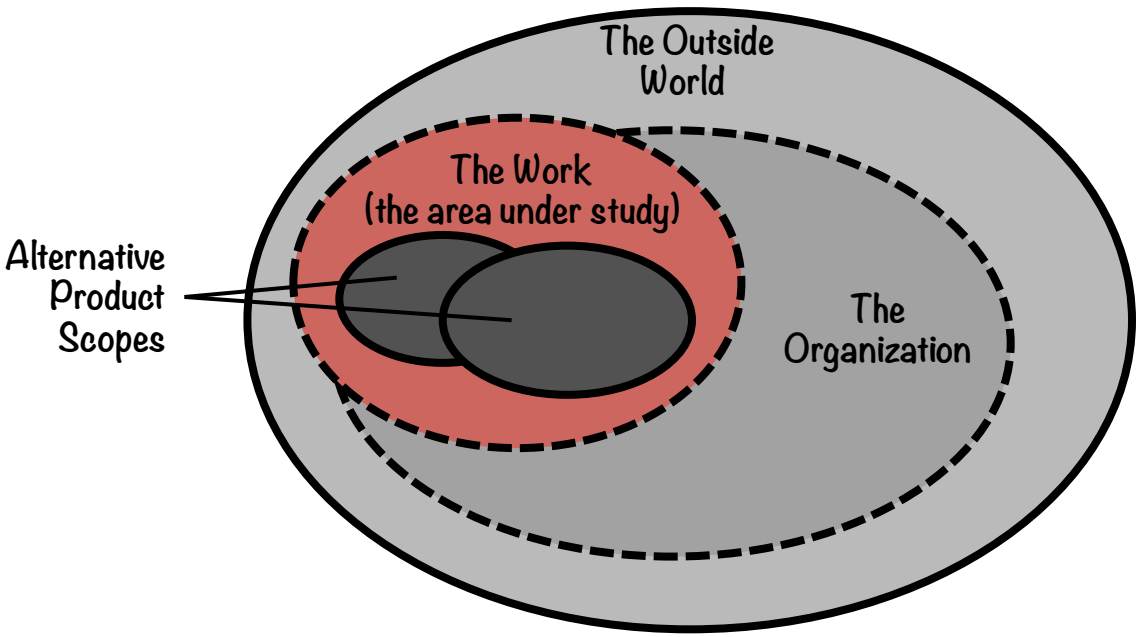
Business Use Cases – decompose the work into work-pieces to manage complexity

Use Case Models – express business use cases in a manner that is easy for all stakeholders to visualize

- Big picture view of the problem

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Understanding the Work



Robertson, Robertson, *Mastering the Requirements Process,* 2012, Figure 3.3.

Start by understanding the work that the owner does (or wants to do).

Have to understand the work itself, before determining what kind of product will improve the work.

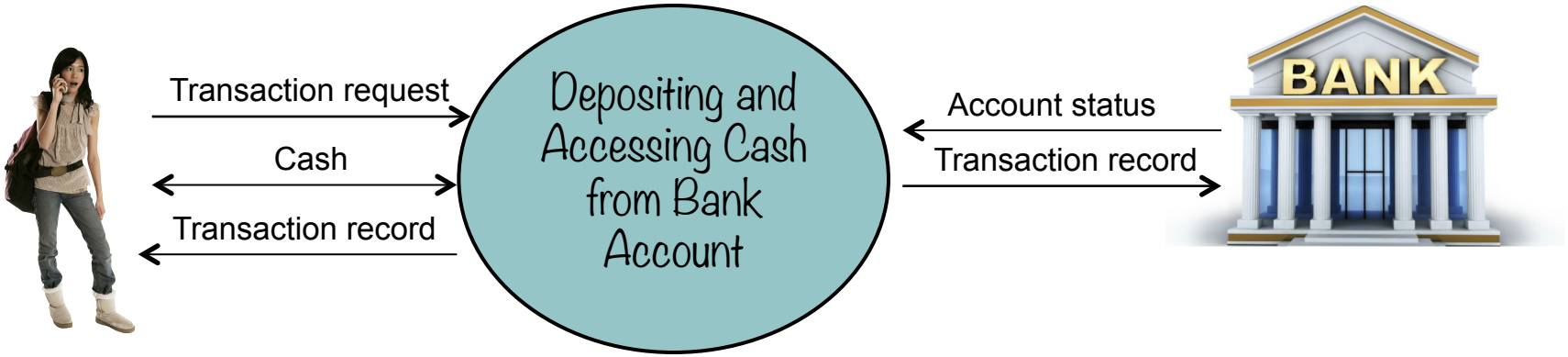
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Understanding the Work

Goal is to understand *what* the existing Work is

- not *how* it is implemented
- focus on externally observable behaviour
- abstract inputs and outputs

Example: Context diagram for Banking

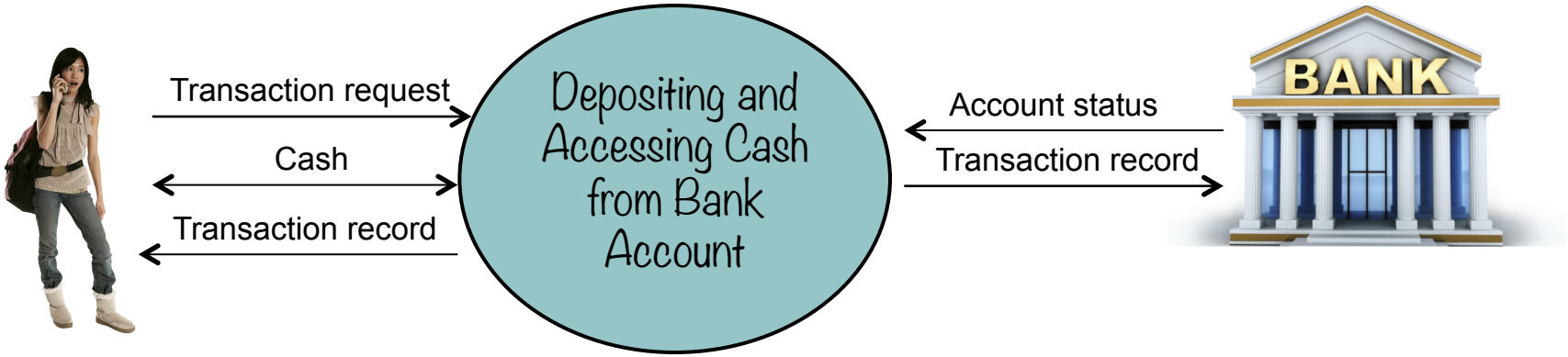


Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Scope of Study

The Work to be studied must include

- anything that you are permitted to change
- anything you need to understand to decide what to change
- anything that can be affected by your product.



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Business Use Cases

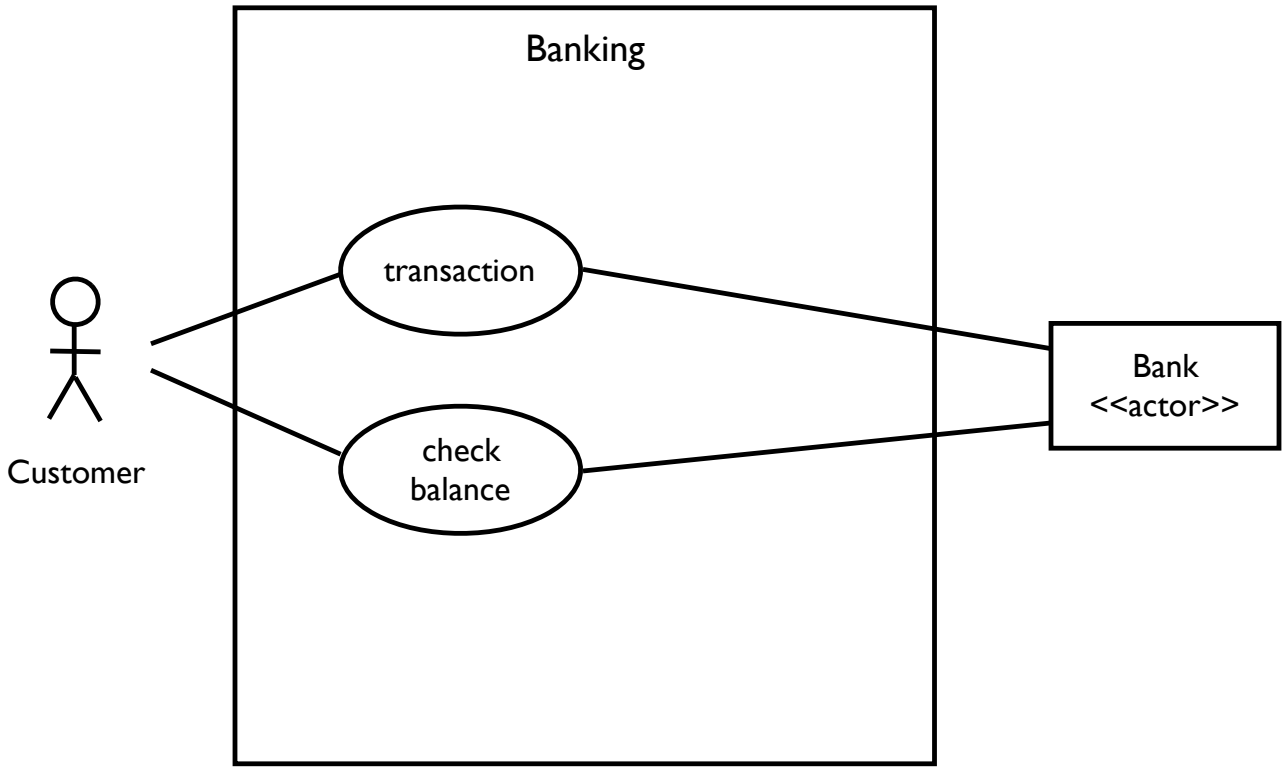
Decompose The Work into vertical slices to reduce complexity.

Each slice is called a use case

- represents some end-to-end functionality
- triggered by an external event (e.g., from adjacent system)
- captures a complete response to a triggering event
- use cases are (ideally) orthogonal to one another

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Business Use Case



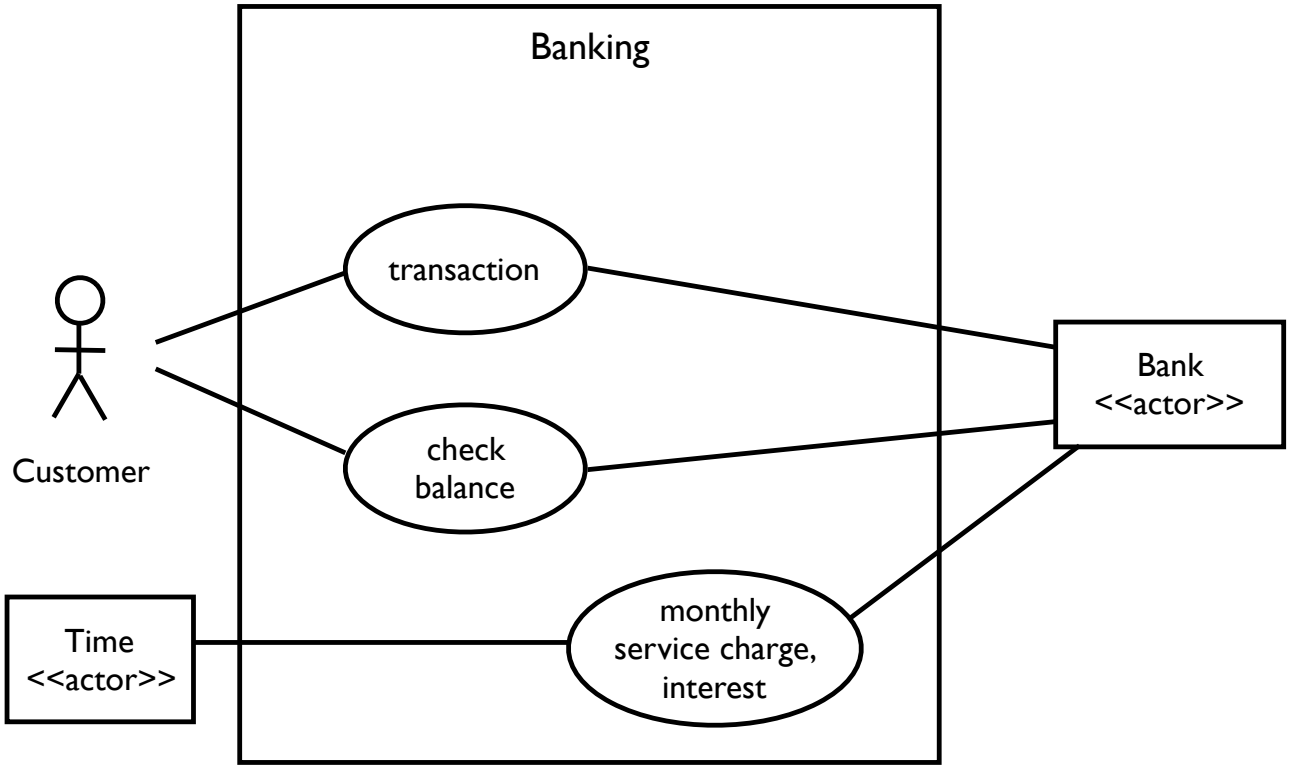
Captures:

- scope of Work
- adjacent systems
- use cases (= business events + responses)
- interactions

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Time-Triggered Use Case

Time-triggered use cases are activated when a date or time comes to pass.



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Park User Fees

Suppose that the city of Waterloo decides to raise funds by instituting users fees for public parks.

Requirements:

R1: Collect $1 fee from each user on entry to the park.

R2: Ensure that anyone who has paid may enter the park.

R3: Ensure that no one may enter park without paying.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Patient Monitoring System

Patients in an intensive-care ward in a hospital are monitored by electronic analog devices attached to their bodies by sensors of various kinds.

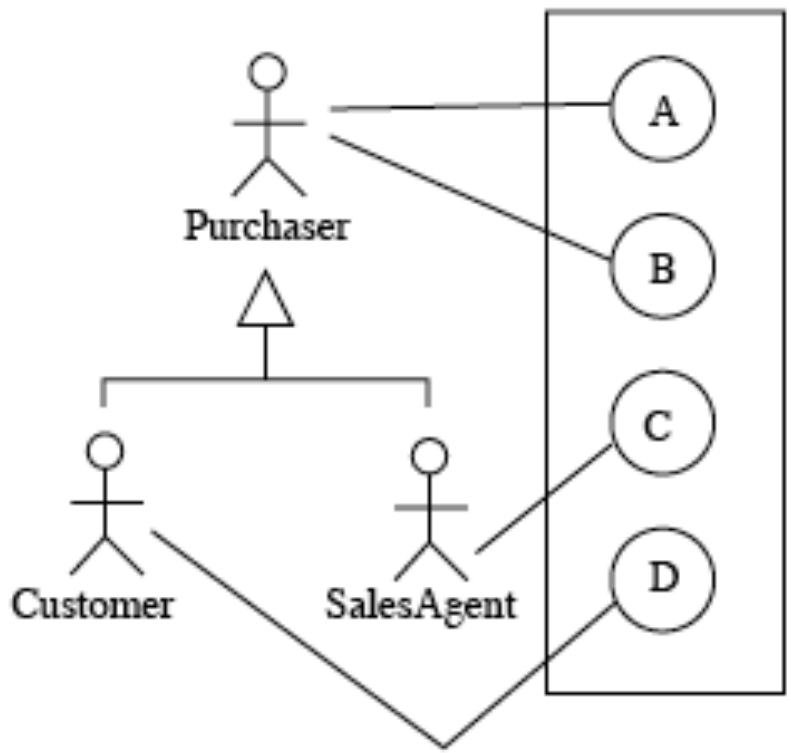
- Through the sensors, the devices measure the patient's vital factors: pulse rate, temperature, blood pressure, and so on.
- A program is needed to read the factors, at a frequency specified for each patient, and store them in a database.
- The factors read are to be compared with safe ranges specified for each patient, and readings that exceed the safe ranges are to be reported by alarm messages displayed on the screen of the nurse's station.

Stevens, Myers, and Constantine, *IBM Systems Journal*, 1974

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Actor Generalization

- Use actor generalization when actors have common *interesting* behaviour
  - i.e., they interact with many of the same use cases
- Factor out common behaviour as an abstract actor
  - Children inherit all relationships with use cases of the parent



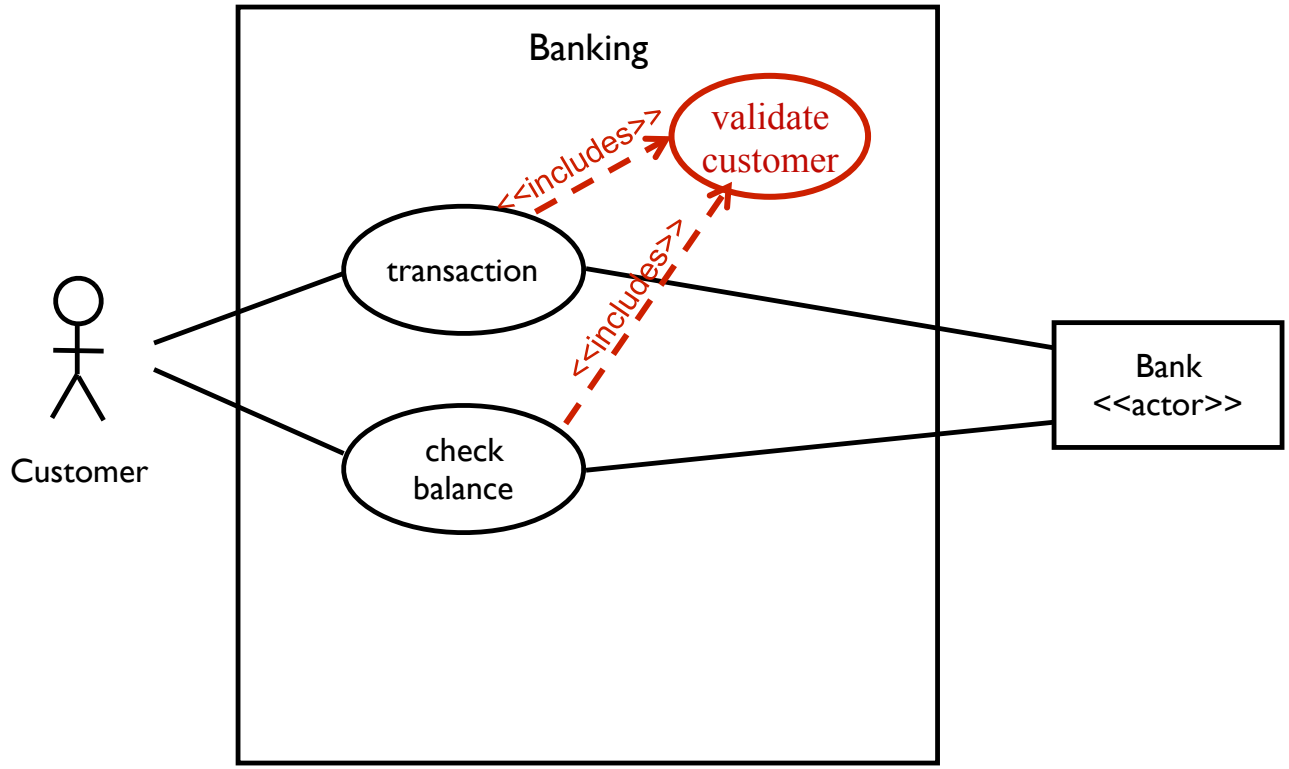
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

«include»

- <<include>> - a sub use case that is used within multiple other use cases (like a procedure call)
  - Avoids repetition of the same steps in multiple use cases, improving readability
  - Specify point of inclusion in base use case
  - When sub use case completes, control returns to base use case

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example



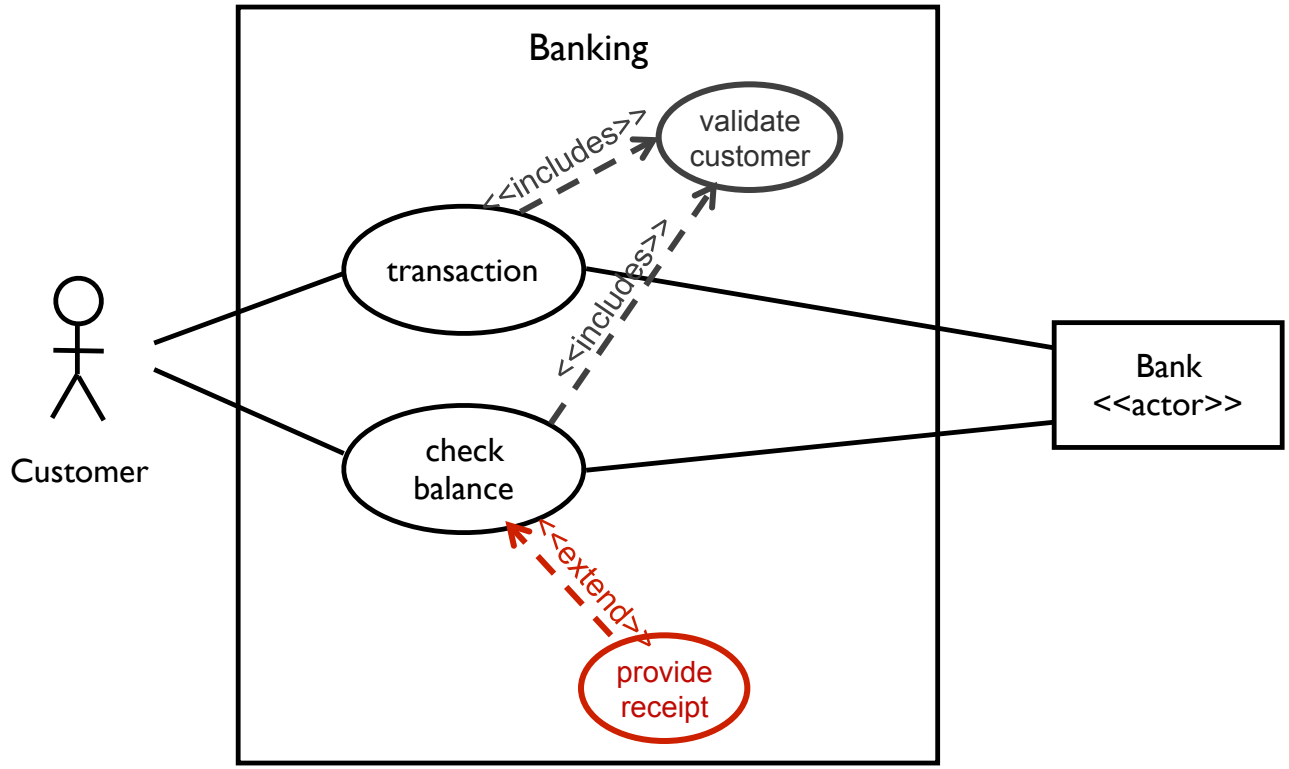
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

«extend»

- <<extend>> - a sub use case that extends or replaces the end of an existing use case
  - Base use case has hooks where it can be extended
  - Unlike «include», base case is complete without extension use case

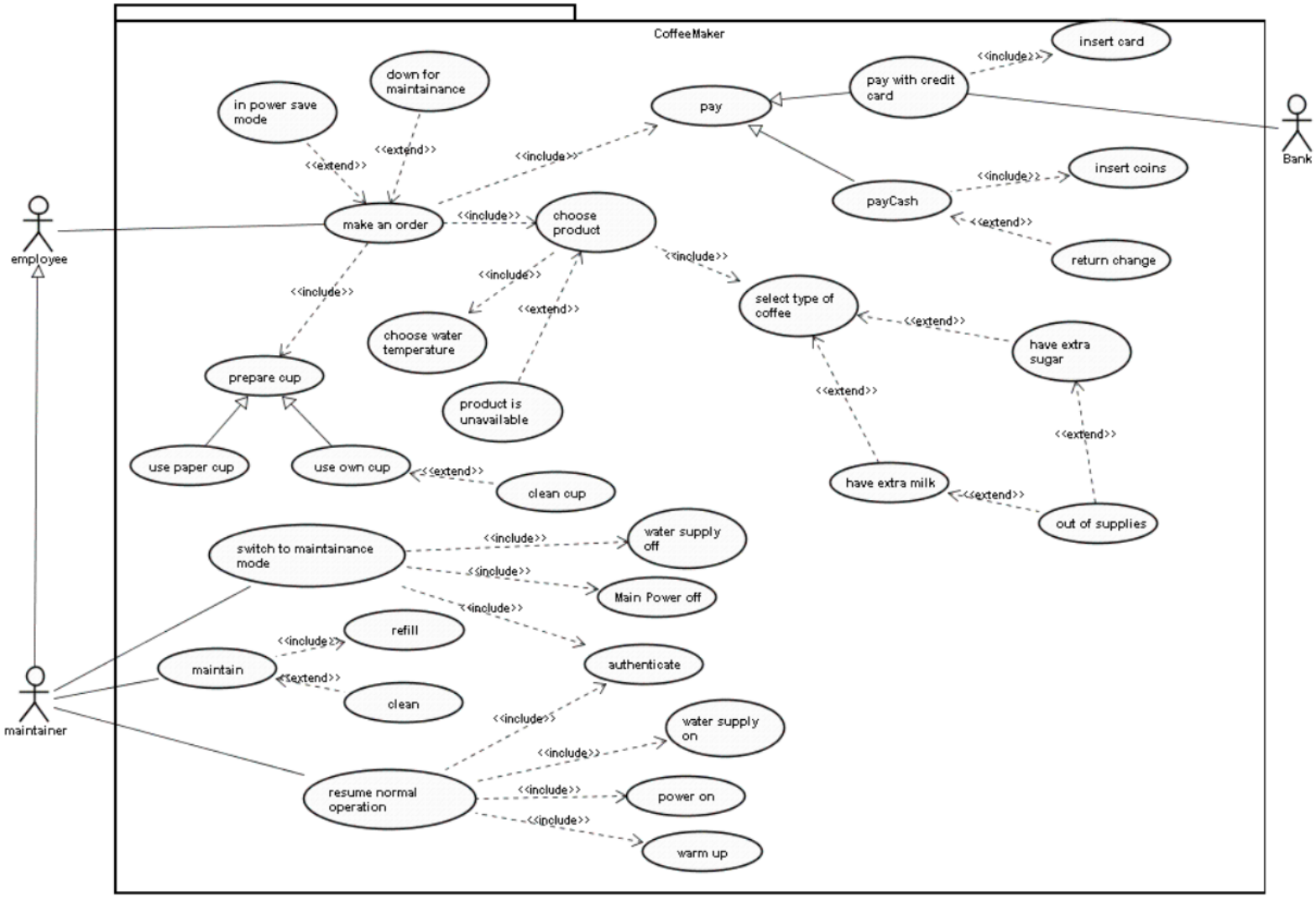
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Bad Example (from the Web)



http://www.iai.uni-bonn.de/III/lehre/vorlesungen/SWT/OOSC06/exercises/exercise2.html

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Before eliciting the requirements of the future system, one must

- Understand the current Work (to be improved)
- Decompose the work into business use cases (to reduce complexity)
- Capture all business use cases in a comprehensive Use Case Model

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Domain Models

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Requirements Models

Model: a simplified version of something complex used in analyzing and solving problems or making predictions

Uses of Models in Requirements

- can guide elicitation
- can provide a measure of progress
- can help to uncover problems, especially omissions
- can help us check our understanding

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Unified Modeling Language (UML)

Structural

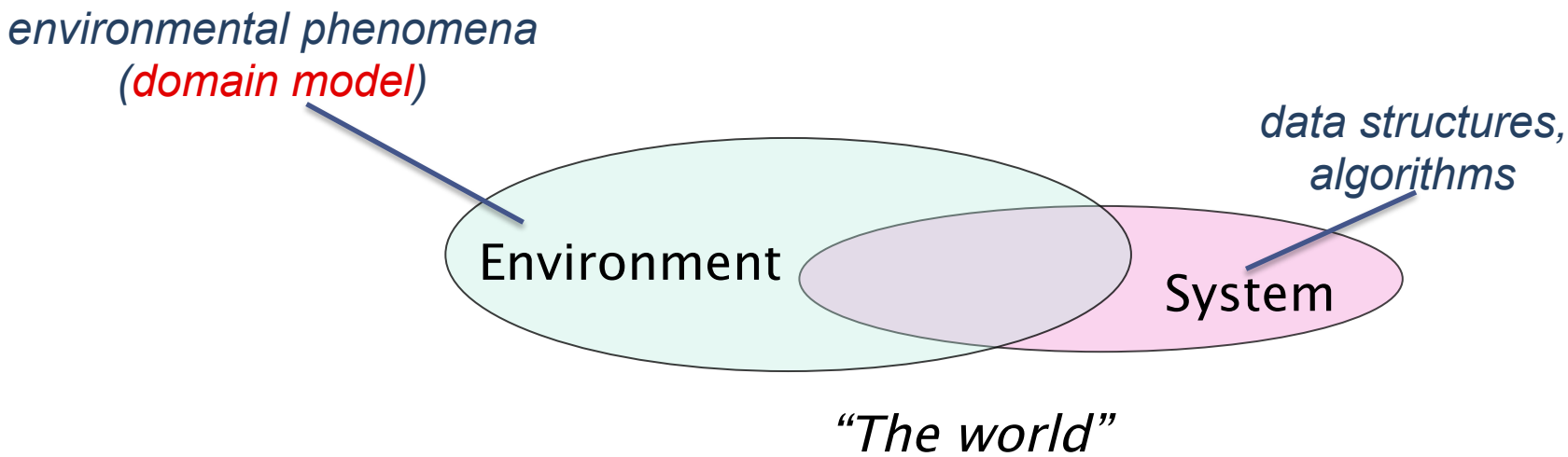
- Class Diagrams
- Object Diagrams
- Composite Structure Diagrams
- Component Diagrams
- Package Diagrams
- Deployment Diagrams

Behavioural

- Use Case Diagrams
- Activity Diagrams
- Interaction Overview Diagrams
- Sequence Diagrams
- Communication Diagrams
- Timing Diagrams
- State Machine Diagrams

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Environmental Phenomena



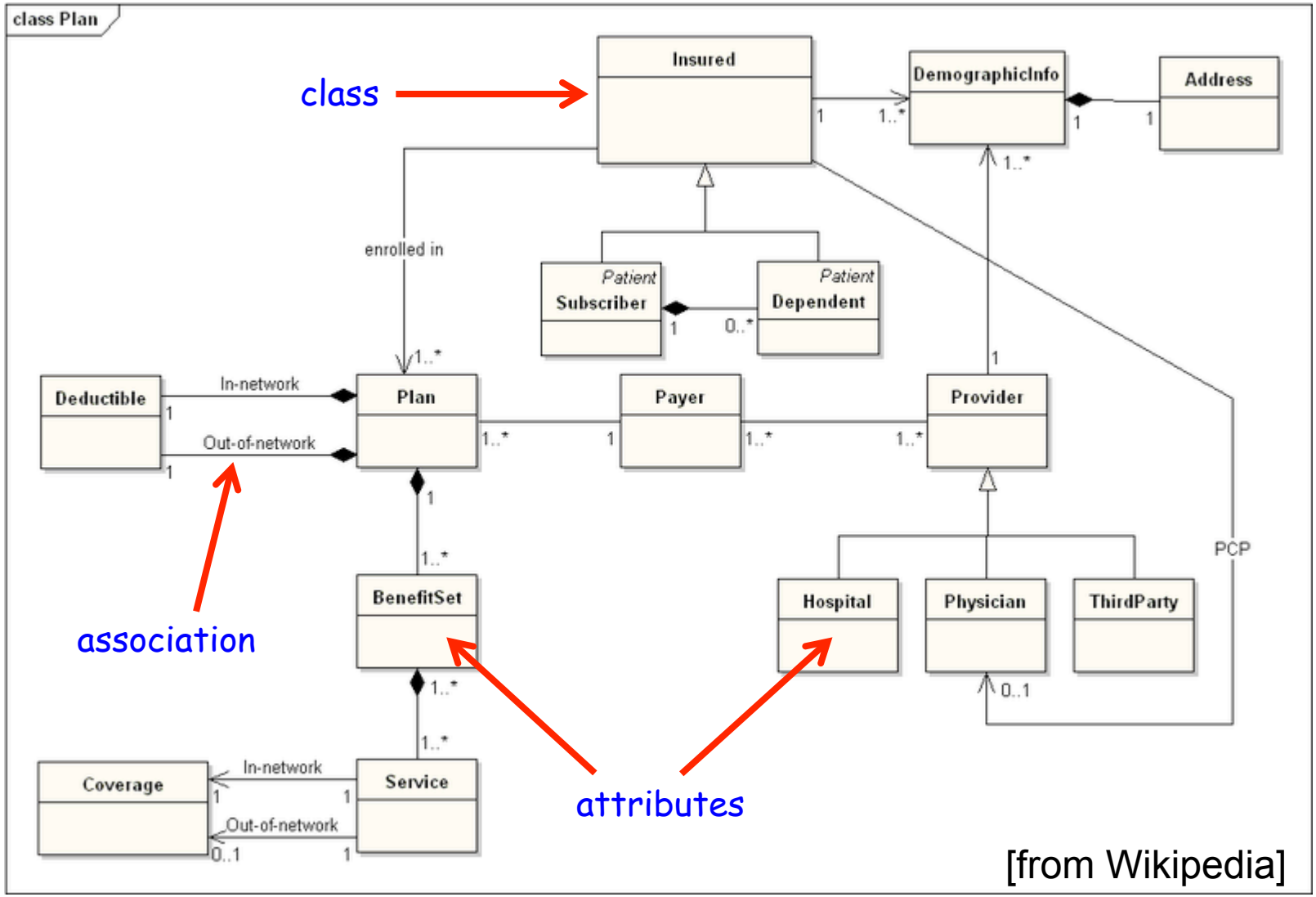
Requirements are expressed in terms of environmental phenomena.

A domain model visualizes the elements of the environment that are relevant to the Work or SUD.

- a graphical data dictionary

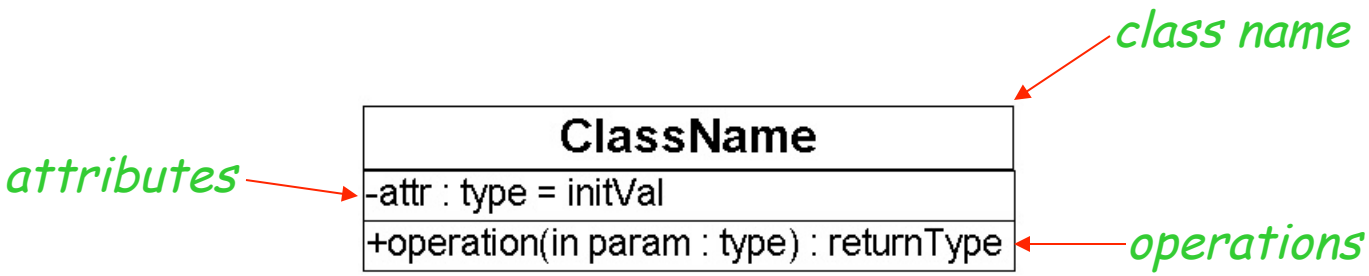
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example of (early) domain model



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Domain Model Classes



Often leave out details early on:

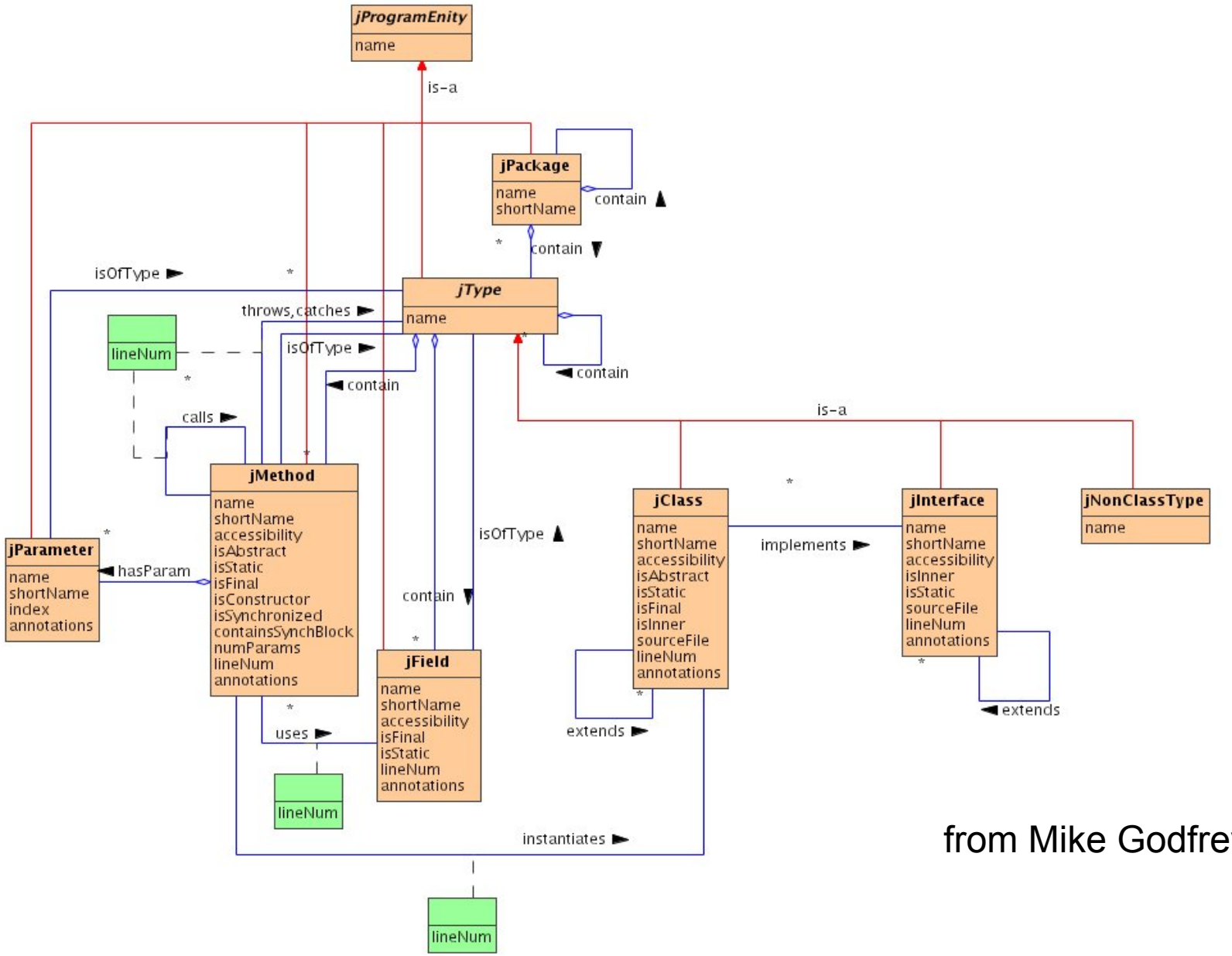
- start with just a class name
- add attribute names
- (maybe) add data types, params, initVals, *etc.*

reqs

- NO operation names
- NO prog. lang. datatypes, access permissions, *etc.:*
  - + `public`, # `protected`, - `private`
  - $ static, / derived

design

Partial domain model for a Java code analyzer



from Mike Godfrey

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Larman's canonical conceptual classes

Conceptual class category	*Example (airline reservation)*
Business transactions	*FlightReservation*
Transaction line items	*SalesLineItem*
Product/service related to a trans. or TLI	*Flight, seat, meal*
Where is the trans recorded?	*FlightManifest*
Roles of people/orgs related to trans; actors in use cases	*Passenger, Airline*
Place of transaction / service	*Airplane, Plane, Seat*
Events (that have noteworthy time/place)	*FlightDepature*
Physical objects	*Airplane*

[Larman Table 9.1]

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Larman's canonical conceptual classes

Conceptual class category	*Example (airline reservation)*
Descriptions of things	*FlightDescription*
Catalogs	*FlightCatalog*
Containers of things (physical or info)	*Airplane*
Things in a container	*Passengers*
Other collaborating systems	*AirTrafficControl*
Records of finance, work, contracts	*MaintenanceLog*
Financial instruments	*TicketCredit*
Schedules, documents, manuals that are regularly referred to when performing work	*RepairSchedule*

[Larman Table 9.1]

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: UW Parking Permits

- All students, employees, and visitors to the university must pay for parking. Students and employees may buy permits that allow them to park in particular lots. Or they may park as visitors, and pay a fee to park in visitors' parking.
  - **Reserved Employee Parking Lots (Lots A, B, H, K, L, O, R)** Each permit is associated with a particular parking lot and allows the holder of the permit to park in that lot, anytime between 6:00AM and 3:00AM. Employees buy a long-term permit for which they are charged a monthly fee. The monthly fee is the same for all reserved parking lots. A permit has a start date and an open-ended end date; the permit ends when the permit is cancelled. Reserved Employee Parking Lots can be used by visitors during the evenings; the starting time for visitor parking is different for each lot.
  - **Unreserved Student Parking Lots (Lots C, N, W, X)** Permits for these lots are sold to students on a per term basis. Holders of such permits may park in any of the unreserved student parking lots anytime between 6:00AM and 3:00AM. There is no visitor parking in the unreserved student parking lots.
  - **Visitor Parking Lots (Y, Z)** There are two parking lots that are dedicated to visitor parking. Anyone can pay a daily fee to park in the visitor lots anytime between 6:00AM and 3:00AM.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

How to find domain entities

Larman recommends three approaches:

1) *Refine an existing domain model*
   - Created for a previous system, or a well understood domain

2) *Use a conceptual class category list*
   - Brainstorm about all the *entities* and *entity kinds* that are relevant to the system

3) *Identify noun phrases*
   - … in use cases (and other docs, if available)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Attributes

An *attribute* is a *simple* sub-part of an object.

- Characteristics or properties of object instances
- Initially, an attribute is modelled as just a name
- Or, an attribute may be a more complex type whose structure is unimportant to the problem
- There are some default UML primitive (simple) types.
  - Feel free to invent meaningful primitive types for your specification, such as “money” or “date”.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Associations

- Represent *important* relationships between (instances of) two conceptual classes
  - Physical relationships are good candidates, but need not be named ("has" or "part-of" can be inferred).
  - Transient relationships are bad candidates
    - e.g., "communicates with"
  - Avoid vague relations like "knows about". Be specific or be quiet!
  - Larman: "Reduce visual noise!"
    - Focus on *"We need to remember this fact"* rather than *"Here's the set of all possible things that are true about the domain model"*
- An association name (if exists) is typically a short verb
- The same two classes can be related by multiple associations

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

UW Parking Permits (Revisited)

- All students, employees, and visitors to the university must pay for parking. Students and employees may buy permits that allow them to park in particular lots. Or they may park as visitors, and pay a fee to park in visitors' parking.
  - **Reserved Employee Parking Lots (Lots A, B, H, K, L, O, R)** Each permit is associated with a particular parking lot and allows the holder of the permit to park in that lot, anytime between 6:00AM and 3:00AM. Employees buy a long-term permit for which they are charged a monthly fee. The monthly fee is the same for all reserved parking lots. A permit has a start date and an open-ended end date; the permit ends when the permit is cancelled. Reserved Employee Parking Lots can be used by visitors during the evenings; the starting time for visitor parking is different for each lot.
  - **Unreserved Student Parking Lots (Lots C, N, W, X)** Permits for these lots are sold to students on a per term basis. Holders of such permits may park in any of the unreserved student parking lots anytime between 6:00AM and 3:00AM. There is no visitor parking in the unreserved student parking lots.
  - **Visitor Parking Lots (Y, Z)** There are two parking lots that are dedicated to visitor parking. Anyone can pay a daily fee to park in the visitor lots anytime between 6:00AM and 3:00AM.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Roles and multiplicities

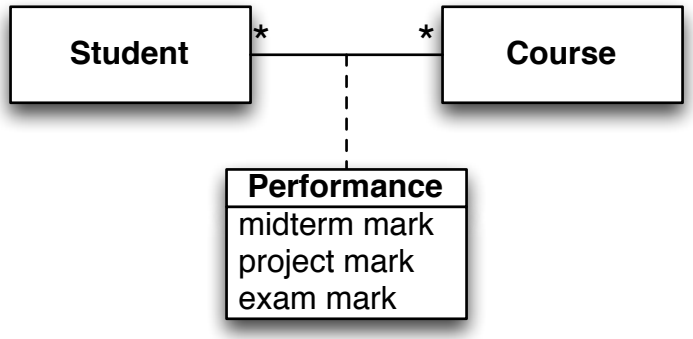
- A *role* is one end of an association
  - It describes how the “local” class is viewed by the “remote” class within the context of the relationship
- Can specify *multiplicities* of associations
  - *e.g.,* 7, 1..52, 0..1, *
  - If not stated, assume that multiplicity is “unstated”
- Roles and multiplicities are optional
- NO *navigability* arrow (this is a design concept)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Association classes

If a relationship has interesting attributes of its own, it should be an *association class*.

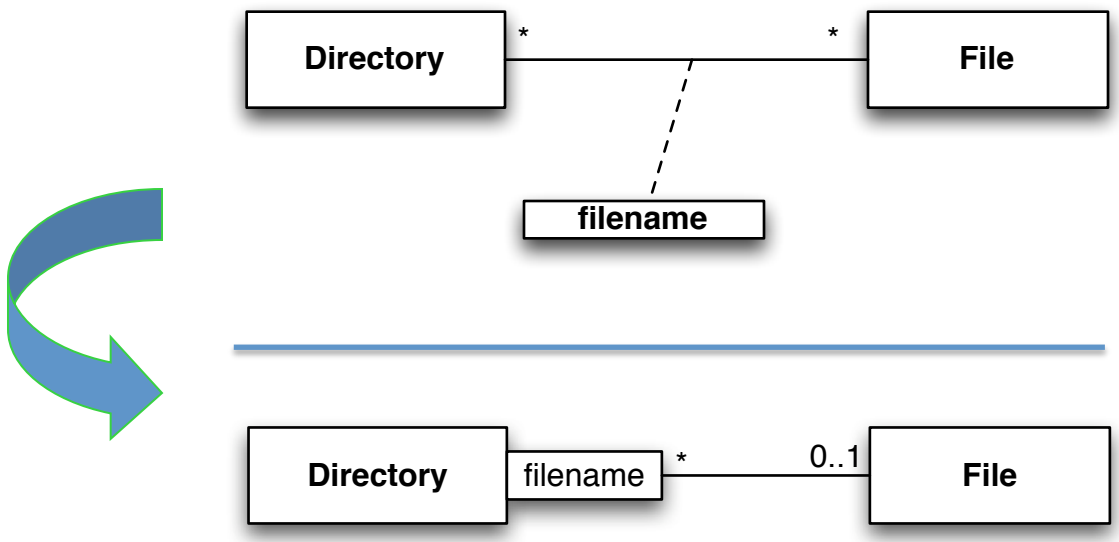
- The state of an association class is knowable only by instances of the two classes
  - If others need to know this info too, just make it into a normal class
- Any two objects can have at most one instance of a given association class



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

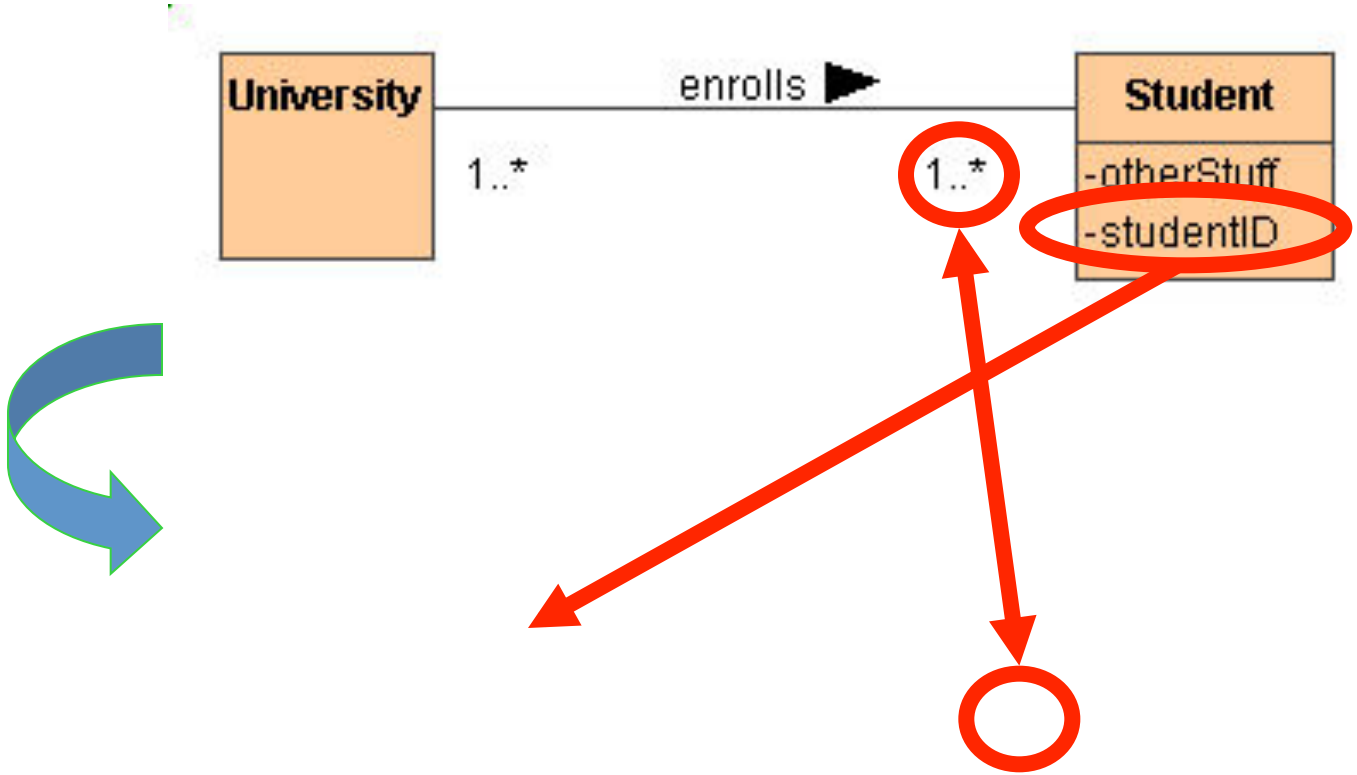
Association qualifiers

- A *qualifier* is a special association attribute used at one end of an association to distinguish among the set of objects at the other end of the assoc
  - *"uniquely identifies"*



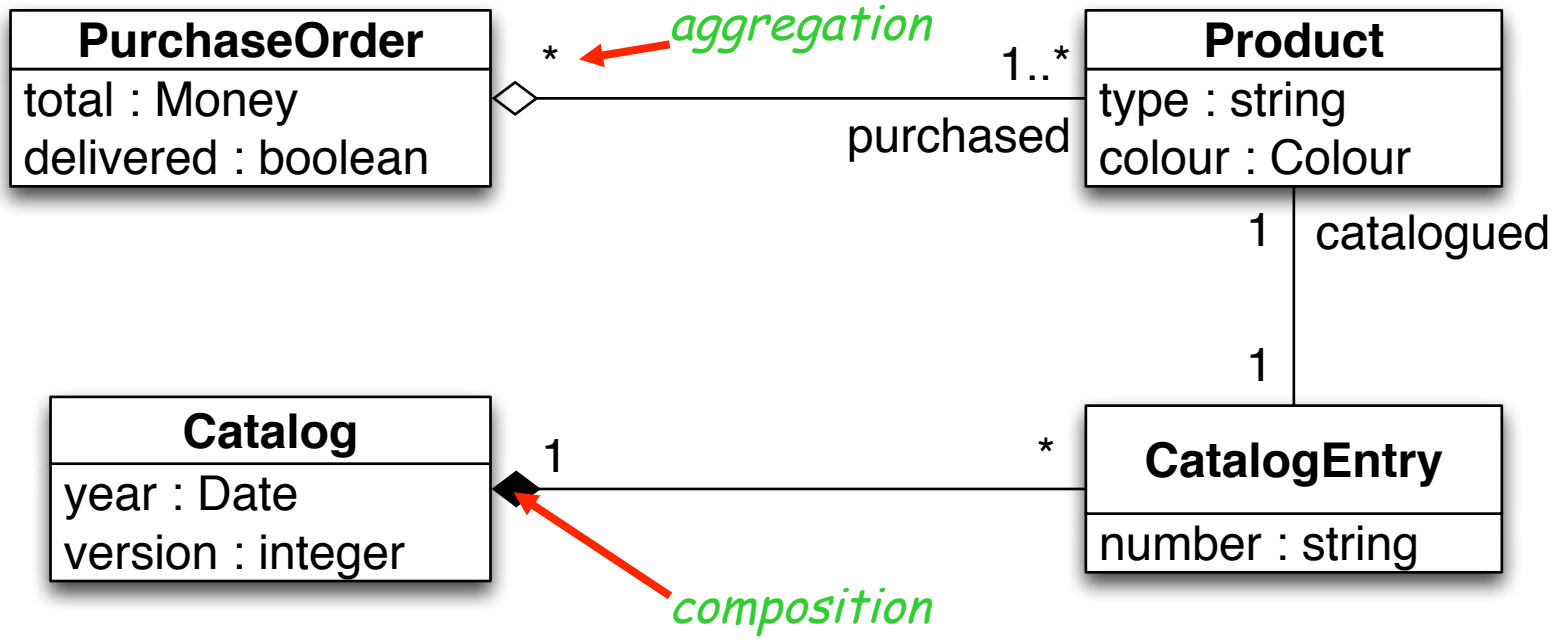
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Association qualifiers



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Composition / aggregation

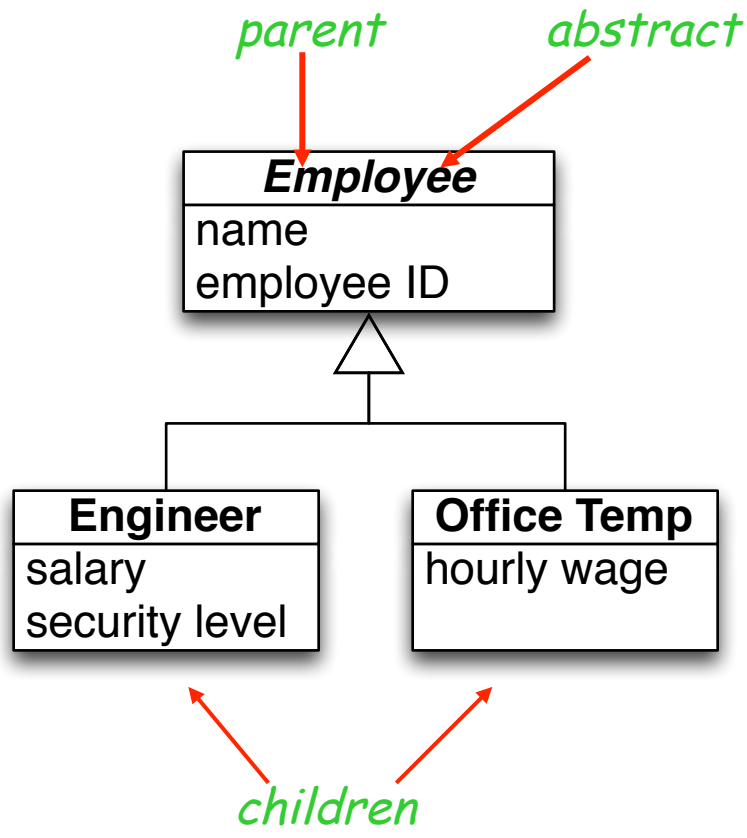


- Composition ◆ an object may *strongly* be a part of at most one container
  - When container dies, subparts die too.
- Aggregation ◇ an object may belong *weakly* to several containers

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Inheritance / Generalization

- The *is-a* (is-a-kind-of) relationship
  - *Engineer* is a kind of *Employee*
- All attributes / associations / other properties of parent are inherited by children
- Children can
  - Fill in abstract details of parent's features
  - Add new features
- Non-leaf nodes can be *abstract*
  - *i.e.,* represent abstractions, not concrete entities
  - Indicated by *italicized* name



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

General advice on domain models

- Class diagrams vs. domain models
  - Making class diagrams is easy: follow the rules
  - Domain modelling is much harder
    - Modelling experience, heuristics
    - Domain knowledge

- Simplicity is your friend
  - Start with just classes and associations
  - Limit the amount of detail you add
    - "I need this info to handle the problem I'm working on"

    c.f., laying out a use case in full detail

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE1 domain models *should* have …

- Attributes and their types
- Multiplicities on all associations (including “1” multiplicities)
- Association names, or role names, for all non-trivial associations
- Qualifiers, to simplify multiplicities on associations
- Actors shown as stick figures or as classes with «actor» stereotype
  - This requires you to show multiplicities between actors and classes, which can be a valuable requirements detail.
- For specifications: Interface devices as classes with «interface» stereotype

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE1 domain models should *not* have

- Class-level operations or methods
- Visibility annotations (i.e., private, protected, public)
- Navigability arrows
- Initial attribute values (unless you need them for model correctness)
- Object construction and destruction functions

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Watch your language!

- The goal is to create a *conceptual* model:
  - Models of real-world entities (customers, accounts, bills) and not of system entities (databases, sw components)
- Focus on the information/artifacts that the system will input, transform, analyze, display, etc; physical and conceptual

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Modelling Conundrums

Reference:
J. Rumbaugh, *OMT Insights, SIGS* Books, Inc., 1996.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Tightly-coupled classes

When there is a one-to-one association between two classes, are the classes distinct or should they be combined into a single class?

Distinct classes:

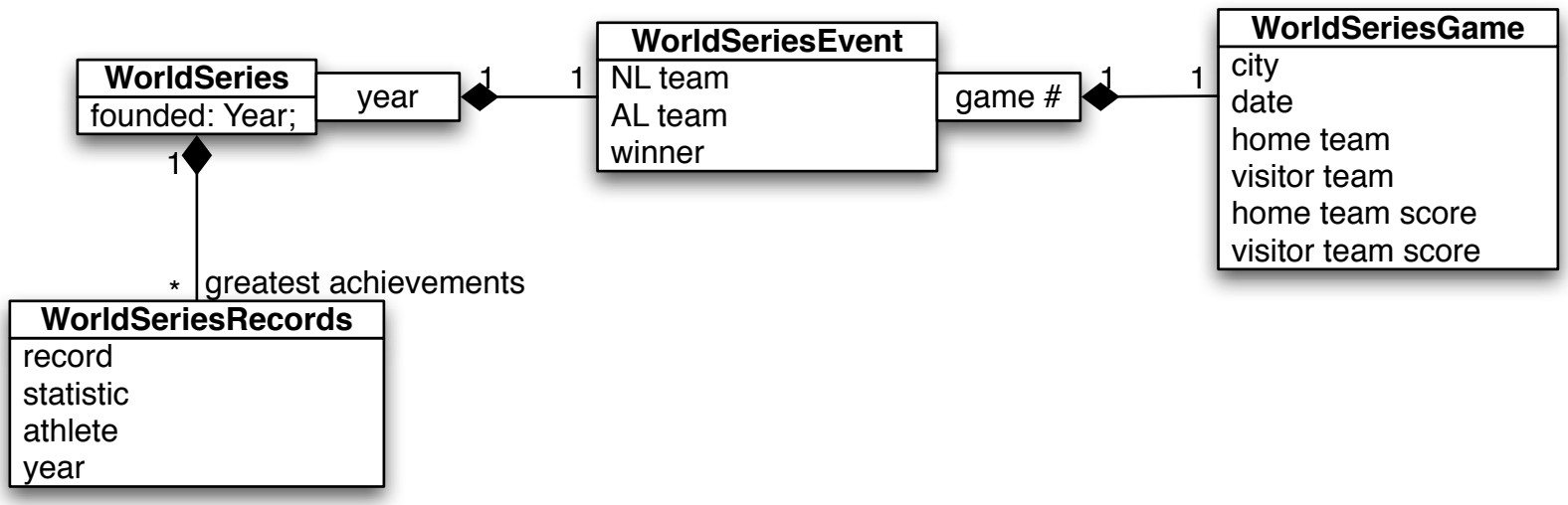
Single class:

If the links are dynamic, then the classes should be distinct

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Similar but distinct classes

A common problem is mixing together two distinct concepts in a single class – especially when the two concepts have the same name.



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Similar but distinct classes

Suppose we want to keep track of cell-phone services, such as text messaging and roaming, which must be subscribed to in advance and that have a charge for each use.

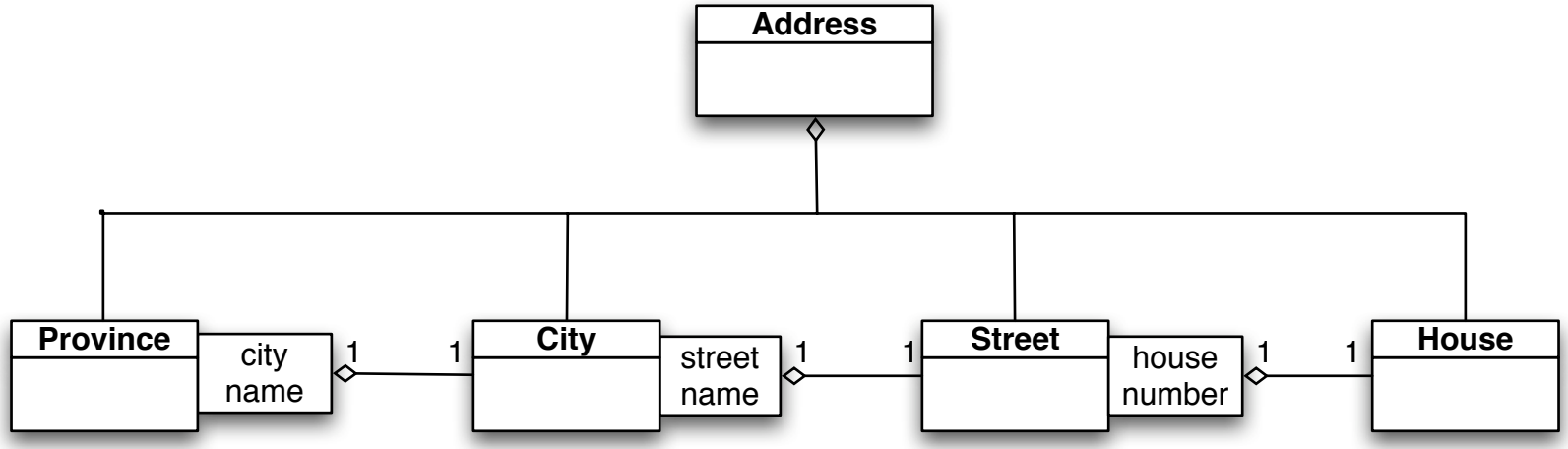
- The phone company offers a set of services to which customers can subscribe.
- A customer may subscribe to one or more services. Each subscription may have properties.
- The use of subscribed services needs to be tracked, so that the customer can be charged for each usage.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Names are not objects

An object has an identity; it can be distinguished from other objects with the same value. A name is simply a string indistinguishable from other strings with the same value.

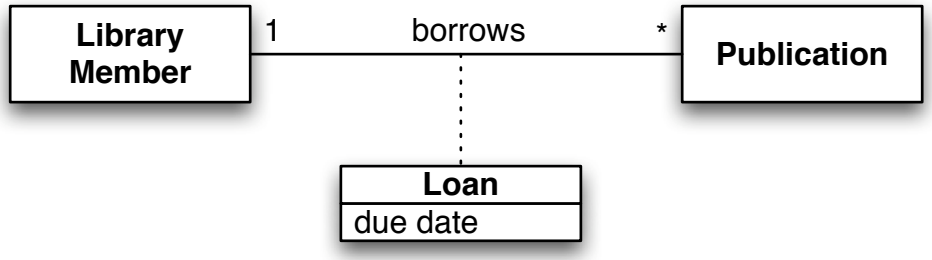
Bad model of address:

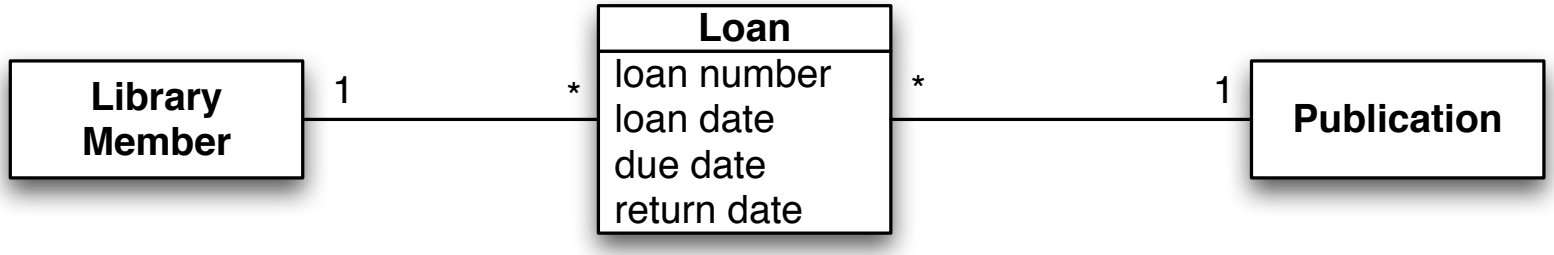


Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Temporal Objects

Sometimes we want to model only the current value of an object, and sometimes we want to record a history of values.





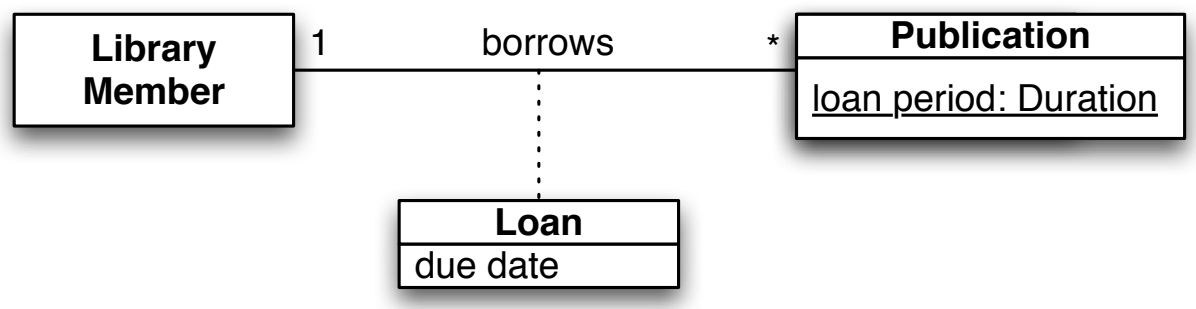
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Class Scope Attributes

By default, every object has its own copy of attributes, and may have its own attribute values.

A class-scope attribute is an attribute whose value is changeable, but is shared by all of the class's object instances. Thus, all objects will have the same class-scope attribute value.

Syntax: underlined attribute declaration



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Class Scope Attributes

What if we wanted to model the case where a publication's loan period depends on the type of the publication (e.g., book, periodical, CD)?

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Class Scope Attributes

What if we wanted to model the case where a publication's loan period depends on a combination of the publication's type and the status of the library member (e.g., faculty, student)?

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Job Mine

Create a domain model for Job Mine. Include

- students
- employers
- jobs
- applications (and their components)
- interview schedules
- rankings
- matches

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Domain models offer a graphical visualization of the environmental phenomena that are relevant to the requirements problem.

- visual data dictionary
- big-picture view of the problem space

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Lightweight Modelling

Readings:

Robertson, S. and Robertson, J., *Mastering the Requirements Process, 3ed.*, Chapter 6 “Scenarios”

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Lightweight Modelling

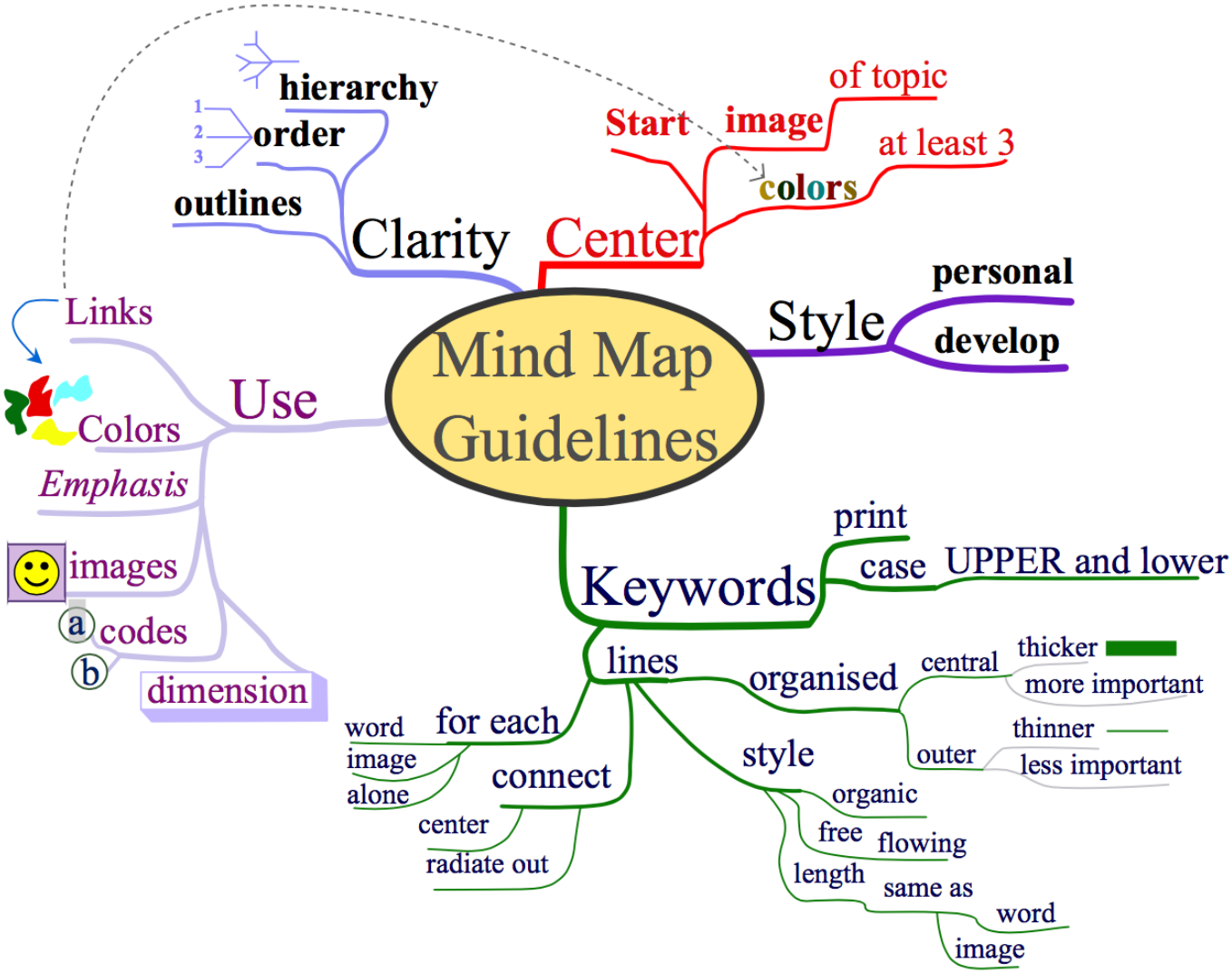
- Mind maps
- Scenarios
- Activity diagrams
- Process models

Act of modelling encourages requirements analyst and stakeholders to take an active role in constructing descriptions

- of current work processes
- of functional requirements

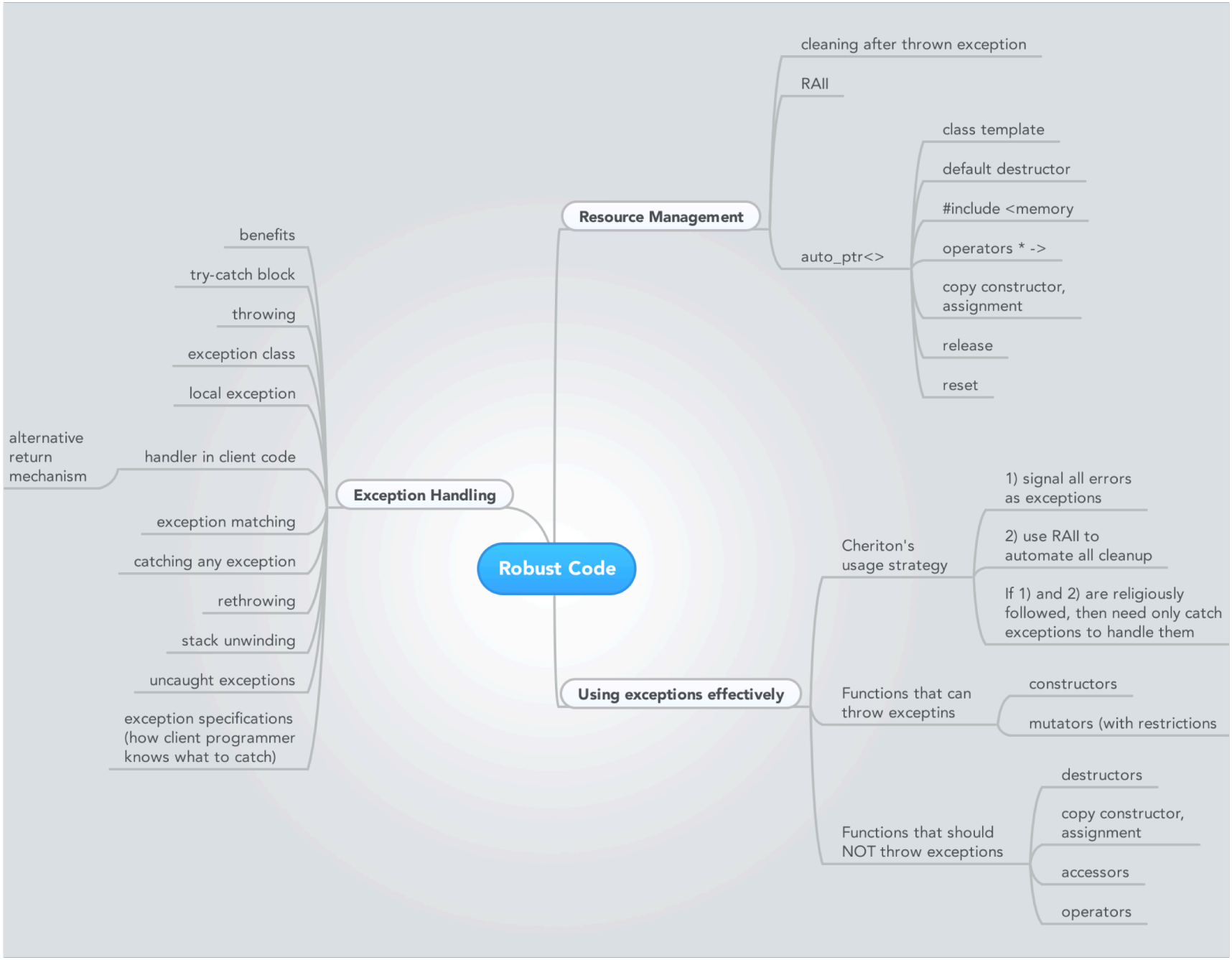
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Mind Maps



https://en.wikipedia.org/wiki/Mind_map

Copyright © Atlee, Berry, Day, Godfrey 1995-2016



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Scenarios

A scenario is one full execution path through a use case, listing only observable actions.

Banking Scenario

1. User requests to withdraw funds, specifies amount
2. Bank authenticates user
3. Bank checks that the account has sufficient funds
4. Bank dispenses cash and receipt
5. Bank records the transaction
6. User takes cash and receipt and leaves

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

More Complex Actions

- If (conditional statement)
- For (iteration expression)
- While (conditional iteration)
- Go To UC*n*

Example:

3. While the Bank checks that account has sufficient funds

3.1 Show advertisements to the customer

These constructs are not needed very often and may be a sign that the scenario is becoming too detailed or too much like pseudo-code.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Variations

- Alternative: a sub use case that achieves the main goal of a use case through a different intended sequence of actions

  Example:

  1. User requests to withdraw funds, specifies amount

  A1.1 User cancels transaction

  6. User takes cash and receipt and leaves

  A6.1 User takes cash and receipt, initiates new transaction

- Exception: a sub use case that captures an unwanted but inevitable deviation

  Example:

  3. Bank checks that account has sufficient funds

  E3.1 Bank cancels withdrawal due to insufficient funds

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Negative Scenarios

- Negative scenario: a scenario that is to be prohibited by the system

Example:

Banking Scenario

1. User requests to withdraw funds, specifies amount
2. Bank authenticates the user
3. Bank checks that the account has *insufficient* funds
4. Bank dispenses cash and receipt

- Misuse case: a scenario that captures undesirable inputs from the user or environment

Example:

2. Bank authenticates user

M2.1 User is an impostor

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Meeting Scheduler

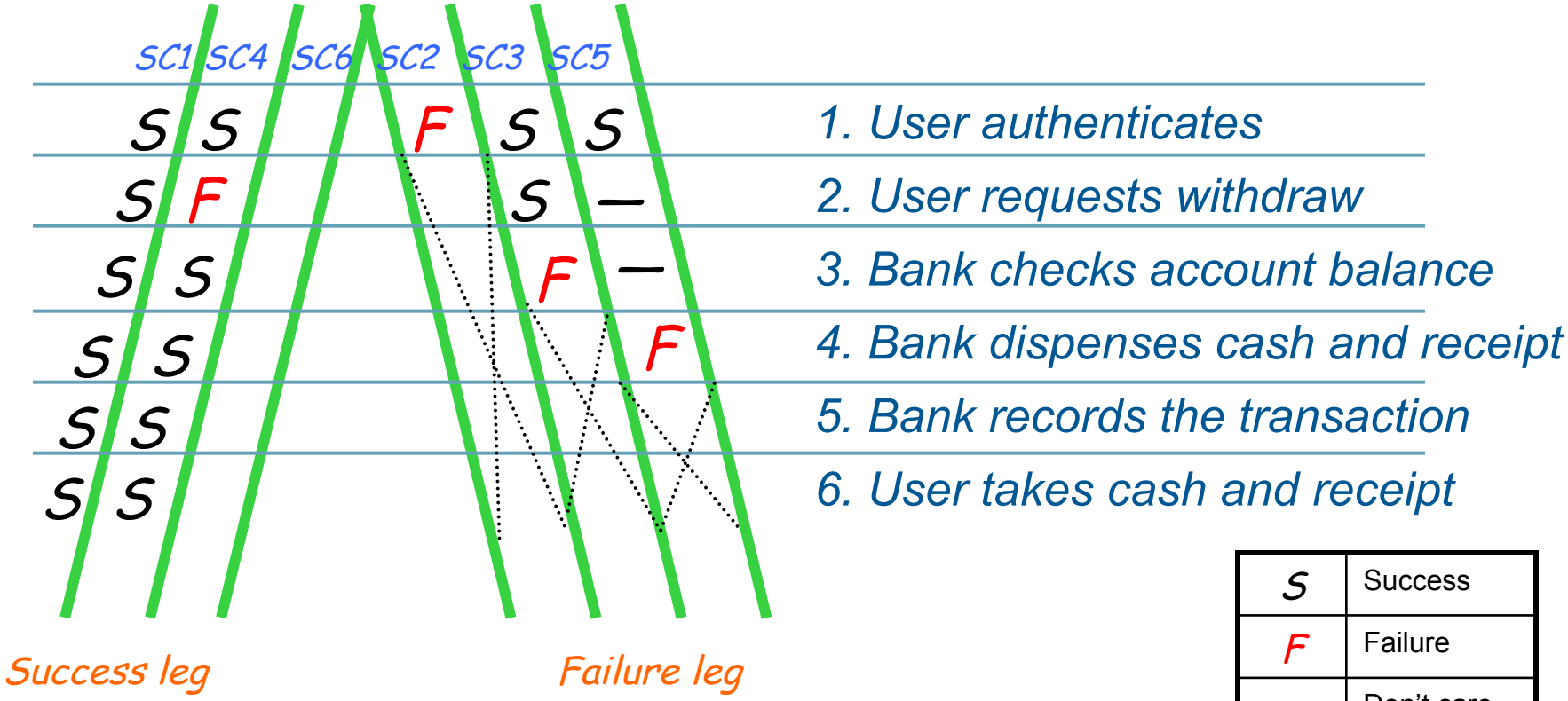
Meeting Scheduler Scenario

1. User requests meeting; specifies participant (important, active, ordinary), agenda, date range
2. Invitations sent to participants, asking about their availability
3. The initiator is asked about room and AV requirements
4. Participants reply with their availability
5. Initiator replies with room and AV requirements
6. AV equipment is reserved
7. When all participants have replied, the system finds a conflict-free date, time and room for meeting
    - 7.1 Room is reserved
8. Confirmation message is sent to all participants and initiator

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Use cases vs. Scenarios

A use case is a collection of success & failure scenarios initiated by an external actor, to achieve a particular goal



S	Success
F	Failure
—	Don't care

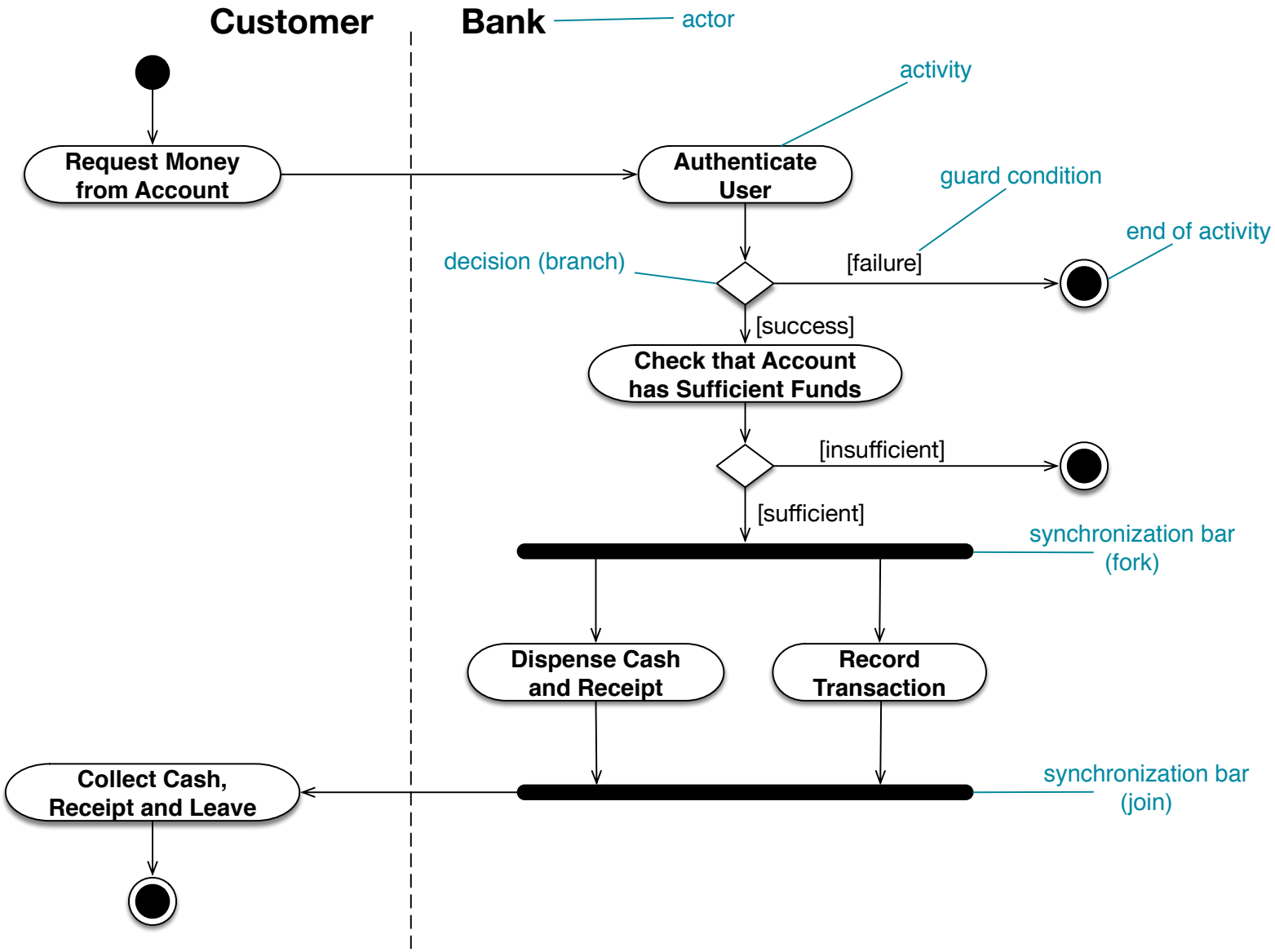
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Template

- **Business Event Name:**
- **Business Use Case Name and Number:**
- **Trigger:**
- **Preconditions:** Sometimes certain conditions must exist before the use case is valid.
- **Interested Stakeholders:**
- **Active Stakeholders:**
- **Normal Case Steps:**
  - Step 1 . . .
  - Step 2 . . .
  - Step 3 . . .
- **Alternatives**:
- **Exceptions**:
- **Outcome**:

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Activity Diagram



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Example: Job Mine

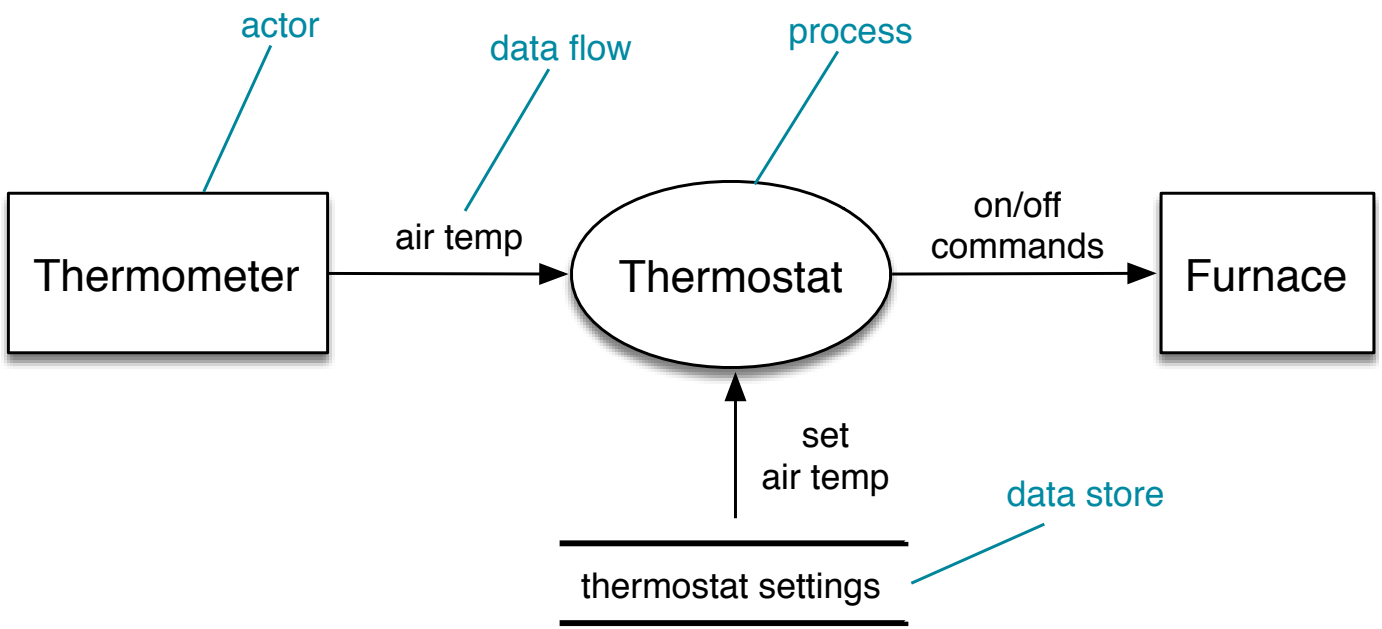
Create an activity diagram to describe the work of a co-op student applying for a job

- Starting with applying for a job
- Ending with a job offer (or not)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Process Model

A process model is a functional decomposition of the work, and the data dependencies between functions.



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Act of modelling encourages requirements analyst and stakeholders to take an active role in constructing descriptions

Want requirements modelling notations to have simple syntax and semantics, for nontechnical stakeholders to be able to read them, understand them, and contribute to their construction

- Mind maps
- Scenarios
- Activity diagrams
- Process models

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Elicitation

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

The Problem Is To Know What the Problem Is

"The most difficult part of requirements gathering is not the act of recording what the user wants, it is the exploratory developmental activity of helping users figure out what they want."

Steve McConnell, *Software Project Survival Guide*, 1998

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Requirements Engineering Process

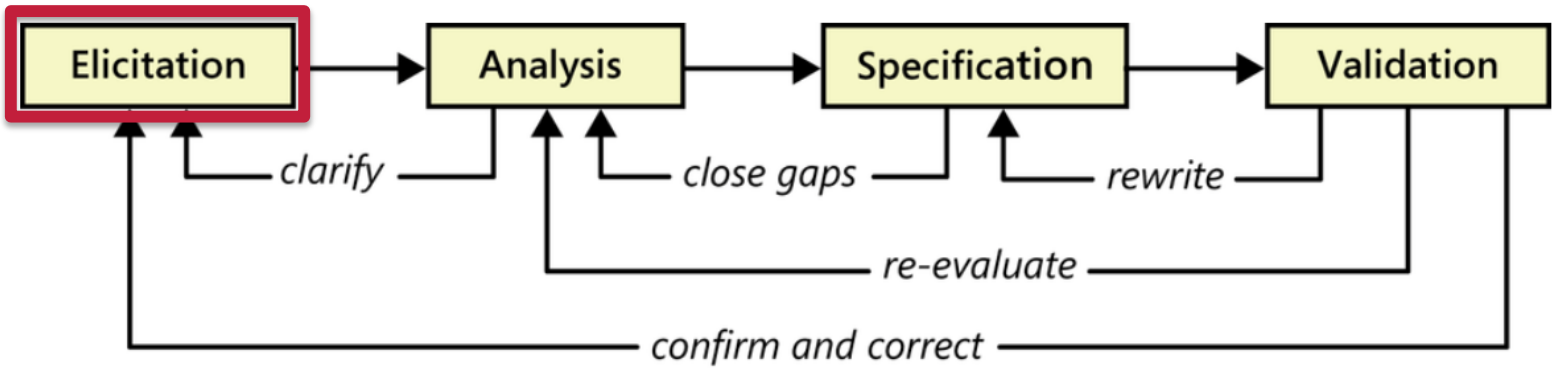


Figure 3-1. Requirements development is an iterative process.

Karl E Wiegers and Joy Beatty, *Software Requirements 3ed*, 2013

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Elicitation

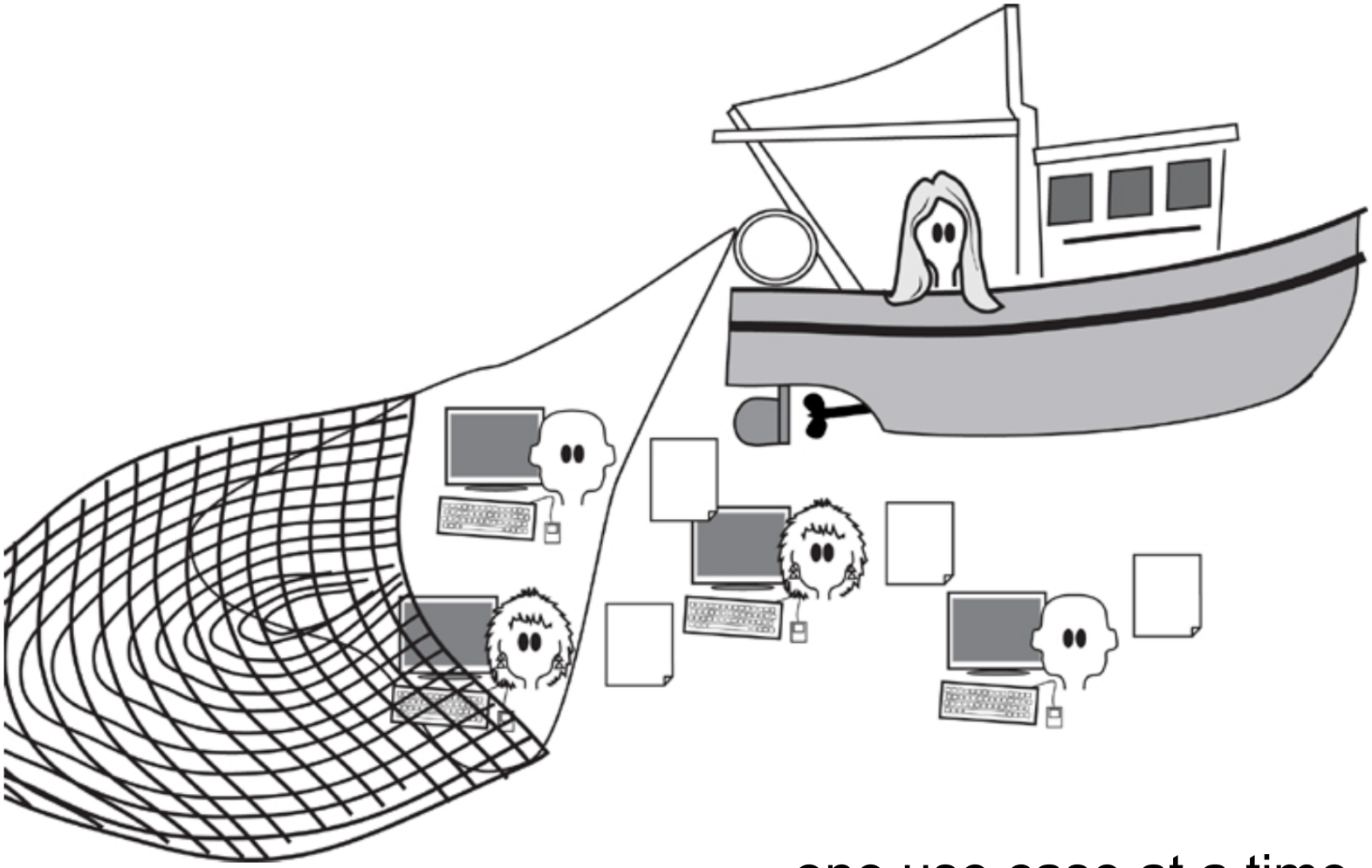
To elicit means “to bring out, to evoke, to call forth”

The purpose of elicitation is to learn the requirements of the system, by

- investigating the current work and current problems
- asking about requirements and goals
- understanding the environment in which the system will operate

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

“Trawling” for Requirements



…one use case at a time.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Information to Elicit

- Required functionality - what the software should do

  record keeping, data computations / transformations, process control, query processing, commands to hardware devices or adjacent systems

- Quality attributes - desired characteristics of the software

  performance, efficiency, safety, security, usability, maintainability, reliability, robustness, availability

- Design constraints - customer-specified limits on solution space

  mandated hardware components, mandated adjacent systems, resource constraints, mandated development process, time/budget constraints

- Environmental assumptions - assumed context of the software

  working status of hardware / software components, assumptions about inputs (data format, rate of input, number of users), operating conditions

- Preferences

  priority rankings of requirements

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Elicitation and Documentation

requirements will be elicited/negotiated, but perhaps not documented; requirements of increments are sketched, with details kept in the heads of the (small group of) stakeholders

because of the number of stakeholders and the length of increments/projects, there is greater need for some documentation – to record information learned, and to communicate it to others

the large number of stakeholders, the length of the project and need to remember information over a long period, the criticality of getting the requirements right – all dictate a strong need for requirements documentation

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Elicitation exercise

DayDreams is a company that helps people to actualize and experience their definition of a perfect day.

- Need volunteer customer

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Elicitation techniques

- Document studies
- Similar companies
- Norms
- Domain analysis
- Requirements taxonomies
- Modelling
- Analysis patterns
- Mockups & prototyping
- Pilot experiments
- Stakeholder analysis
- Questionnaires
- Interviews
- Observation
- Task Demo
- Ask suppliers
- Domain workshop
- Personas
- Systemic Thinking
- Brainstorm
- Creativity workshop

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Artifact-based elicitation

Idea: learn as much as we can by studying documentation, systems, artifacts, etc. before asking for stakeholders' time

- *Document studies*
- *Similar companies*
- *Norms* *Artifact-based*
- *Domain analysis*
- *Requirements taxonomies*
- Modelling
- Analysis patterns
- Mockups & prototyping
- Pilot experiments

- Stakeholder analysis
- Questionnaires
- Interviews
- Observation
- Task Demo
- Ask suppliers
- Domain workshop
- Personas
- Systemic Thinking
- Brainstorm
- Creativity workshop

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Documents

- System documentation

  e.g., existing requirements specifications, design documents, bug reports, change requests user manuals, work procedures, usage statistics, marketing data, performance figures

- Environment documentation

  e.g., organization charts, business plans, policy manuals, financial reports, minutes of important meetings

- Domain analysis

  e.g., textbooks, surveys, standards, regulations, the Web

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Norms

Build a better X.

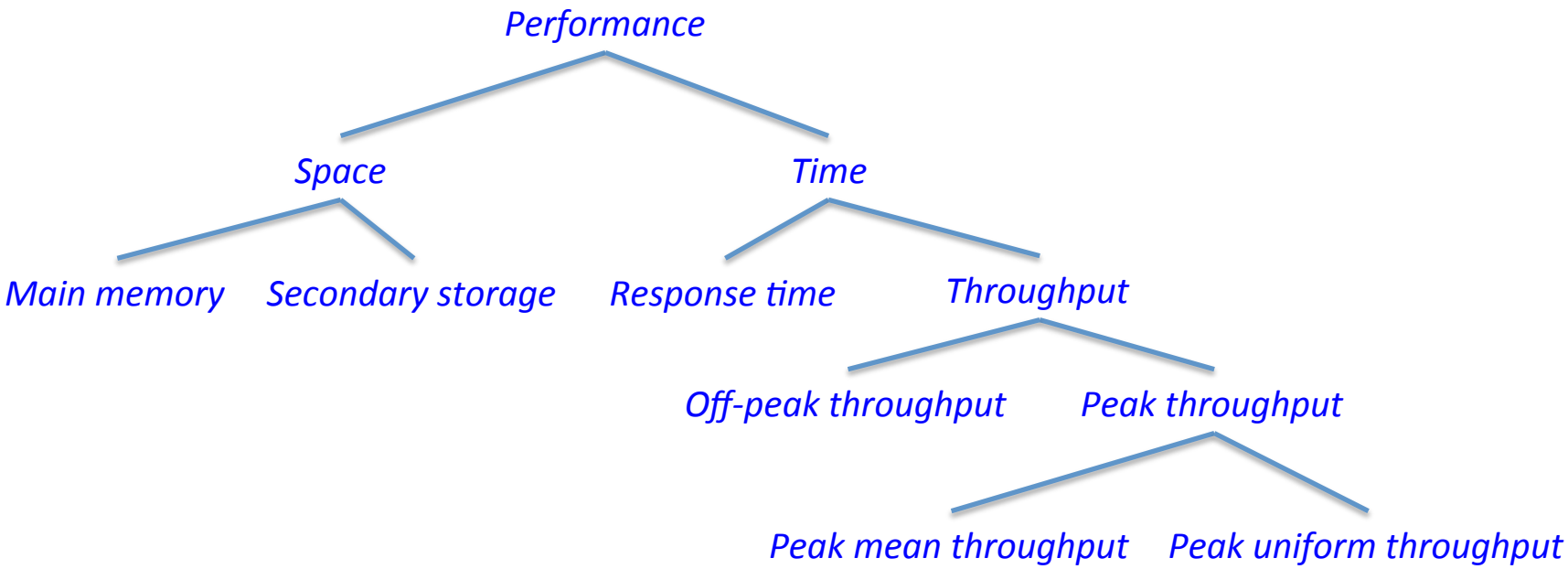
Example: Build a better messaging app that runs on Linux and supports multiple messaging protocols and photos.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Requirements taxonomies

Requirements taxonomy — classification of requirements; the classification can act as a checklist of details to be elicited.

Example: Domain-*independent* taxonomy for performance-related NFRs



Example: Domain-*dependent* taxonomy for information systems:

PIECES ≡ [P]erformance, [I]nformation and data, [E]conomy, [C]ontrol, [E]fficiency, and [S]ervices

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Model-based elicitation

Idea: To re-express the requirements in a different language, which can raise new questions.

- Document studies
- Similar companies
- Norms
- Domain analysis
- Requirements taxonomies
- *Modelling* *Model-based*
- *Analysis patterns*
- *Mockups & prototyping*
- *Pilot experiments*

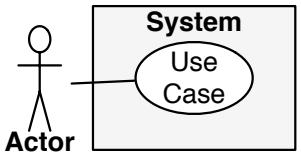
- Stakeholder analysis
- Questionnaires
- Interviews
- Observation
- Task Demo
- Ask suppliers
- Domain workshop
- Personas
- Systemic Thinking
- Brainstorm
- Creativity workshop

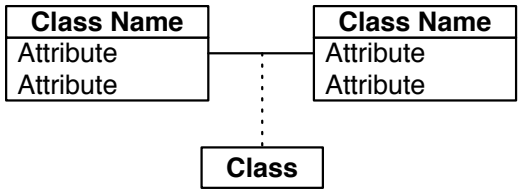
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Models

The act of re-expressing the owner's work or requirements as models in different languages often reveals "holes" in our understanding

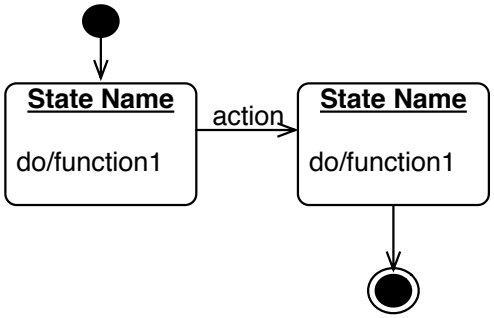
- Ideally, models are simple enough that stakeholders are encouraged to modify them
- Models are useful in requirements documentation






```
context Membership inv
self.transactions->collect(points)->sum = self.points
```

Display Available Numbers	ID: O12	Importance: D
Overview: Display all of the available phone numbers		
Inputs: requestor:Administrator		
Preconditions: requestor has authenticated, there is no data lock on phones (set of Phone)		
Modifies: *none*		
Postconditions: returns all num $\in$ phones.phoneNumber; numbers are displayed in numerical order		
Exceptions: There is a data lock on phones, and error code "operation failed" is returned		
References: Meeting #3: R7		



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Effective Models

Restrict the amount of detail we include in our models of the current system—there is no point modeling every tiny facet of something we are about to replace.

- scenarios
- activity diagrams
- process models
- business rules

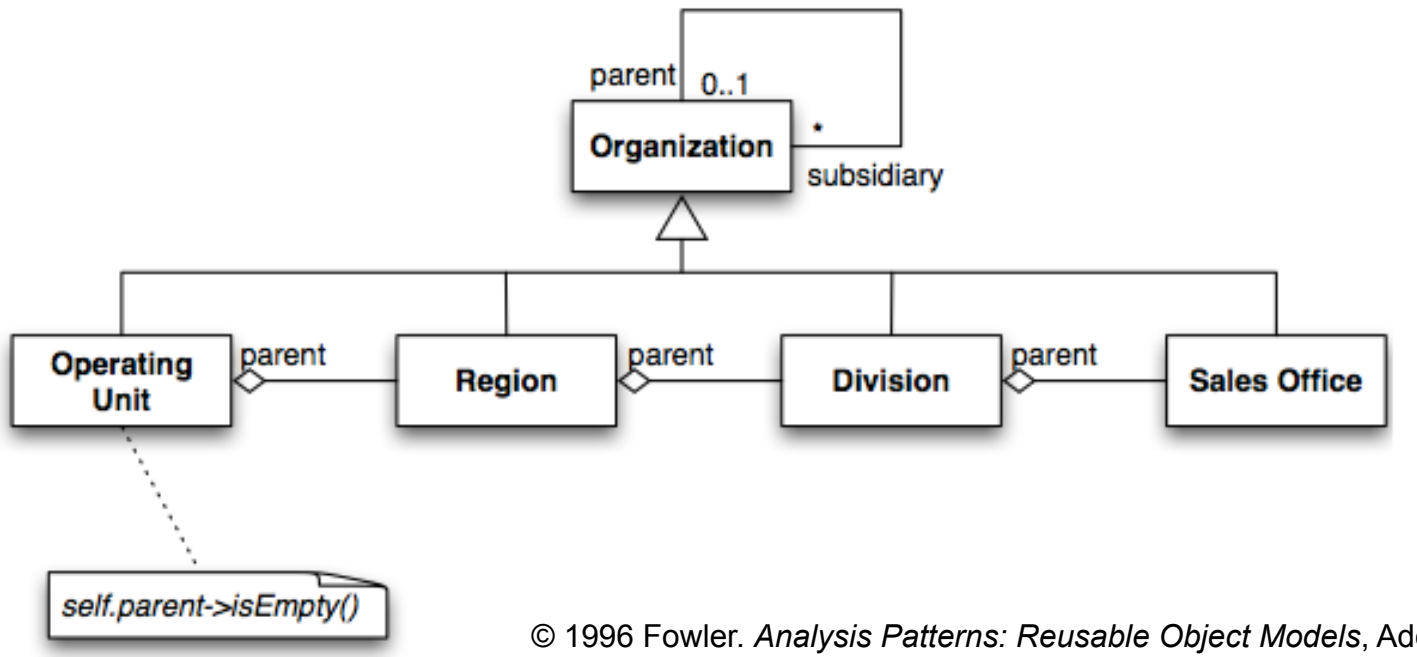
Do not restrict the scope of the work being modelled – the more of the work we study, the more opportunities for improvement that will emerge.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Analysis patterns

Analogous to design patterns, which are templates for OO designs, there is work on analysis patterns, which are templates for modelling common business problems.

e.g., Pattern for organizational relationships

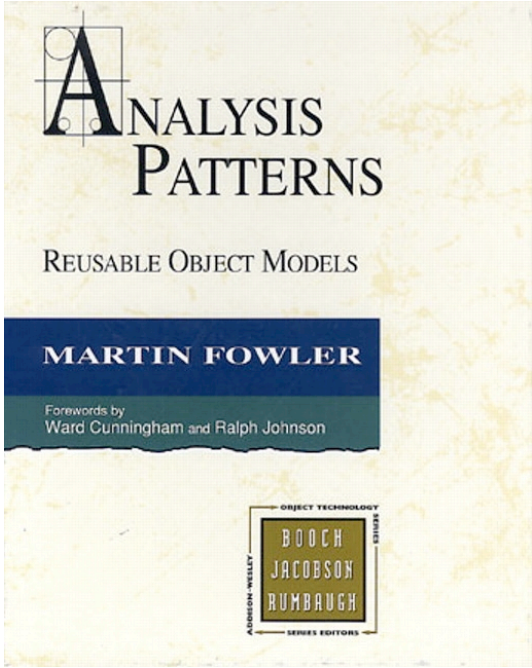


© 1996 Fowler. *Analysis Patterns: Reusable Object Models*, Addison-Wesley

Analysis patterns

The best known collection of analysis patterns is Fowler's *Analysis Patterns: Reusable Object Models.*

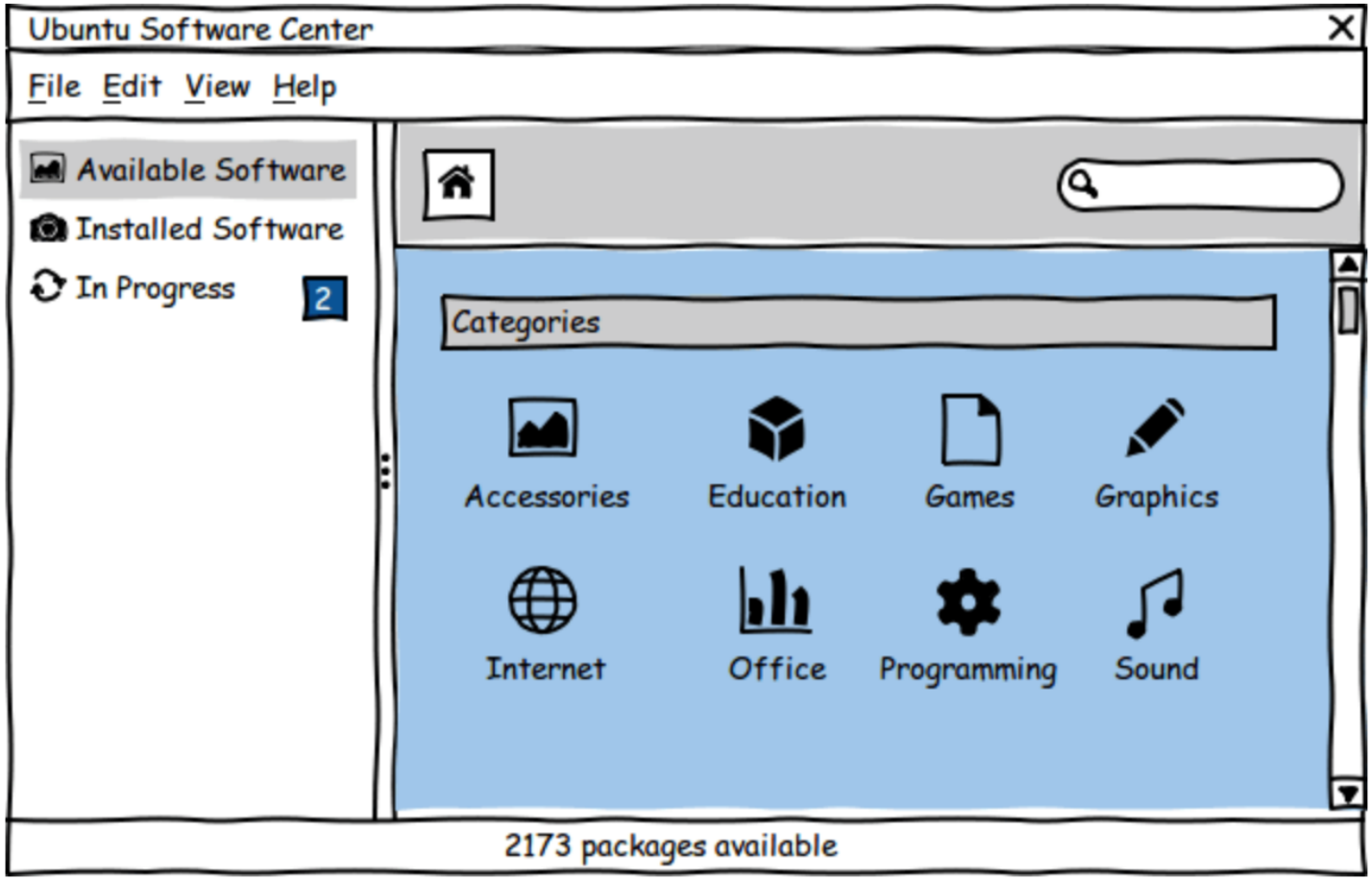
- Organization patterns — about modelling accountability
- Observations and Measurements
- Accounting, Transactions
- Planning
- Trading — financial service systems



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Mockups and Prototypes

Sketch the essence of a solution, and use to bait stakeholders into providing new requirements details



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Stakeholder-based elicitation

Idea: Acquire detailed information about the system-to-be that is problem specific or stakeholder specific.

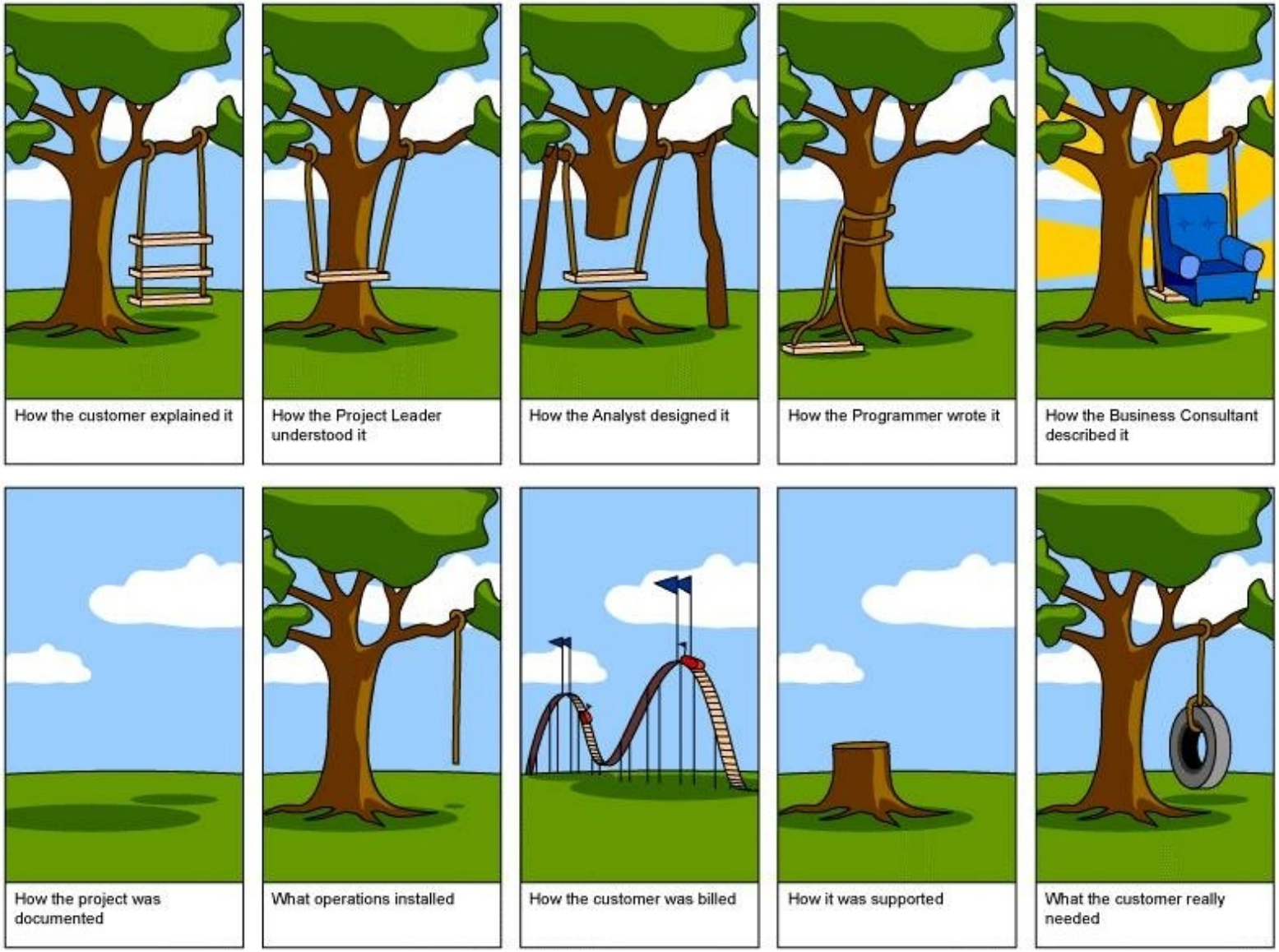
- Document studies
- Similar companies
- Norms
- Domain analysis
- Requirements taxonomies
- Modelling
- Analysis patterns
- Mockups & prototyping
- Pilot experiments

- *Stakeholder analysis*
- *Questionnaires*
- *Interviews*
- *Observation*
- *Task Demo*
- *Ask suppliers*
- *Domain workshop*
- *Personas*

Stakeholder-based

- Systemic Thinking
- Brainstorm
- Creativity workshop

Copyright © Atlee, Berry, Day, Godfrey 1995-2016



Origins unknown. Variations have circulated since the 1960s.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Stakeholder-based elicitation

"You have two ears and one mouth. I suggest that you use them in that proportion."

—G. K. Chesterton

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Analyze current users

- To understand the problem, analyze existing “system” if possible:
  - Questionnaires
  - Interviews
  - Observe current users/apprenticeship

- The goal is to find out:
  - What is used, what isn’t, what’s missing.
  - What works well, what doesn’t.
  - How the system is used, how it was intended to be used, what new ways we want it to be used.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Questionnaires

Questionnaires are useful when information has to be gathered from a large number of people, particularly users.

- Closed questions (to gather opinions)
- Open questions (to gather suggestions)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Interviews

Interviews are useful for asking targeted, stakeholder-specific questions.

- Elicit stakeholder-specific ideas, opinions
- Elicit details that only the stakeholder can answer

Want to phrase questions as open questions, to elicit more details from the stakeholder

- who, what, when, where, ***why***

Good listening skills means focusing on what the stakeholder is actually saying; giving the stakeholder some time to articulate an answer

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Interview Guidelines

- Set the interview in context
- Limit the duration of the interview (60-90 min) and stick to it
- Start with stakeholder background, goals
- Follow with use-case specific questions
  - Appropriate responses to business events
- Feedback your understanding of stakeholder's answers
  - Involve stakeholder in model building
  - Encourage stakeholder to change models
- Use the stakeholder's terminology
- Write down everything you are told
- Thank the stakeholders for their time

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Common interviewing mistakes

- Not interviewing all of the right people.
- Asking direct questions too early.
- Interviewing one-at-a-time instead of in small groups.
- Assuming that stated needs are exactly correct.
- Letting one person dominate a group discussion

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Ethnographic analysis

Ethnographic analysis is direct, first-hand observation of user behaviour

- An attempt to discover the social/human factors in a system.
- Studies have shown that work is often richer and more complex than suggested by simple system models derived by interviews alone.
- Can identify the used and critical existing features
- But focuses on existing solutions

© 2002 Michael Neugebauer

Apprenticeship

- Apprenticing is based on the idea of masters and apprentices
- The apprentice sits with the master craftsman (the user) to learn the job
  - By observation, asking questions, doing some of the job under the master's supervision.
- While working that the user can:
  - describe the task precisely
  - explain why the task is done this way
  - list the exceptions that can occur

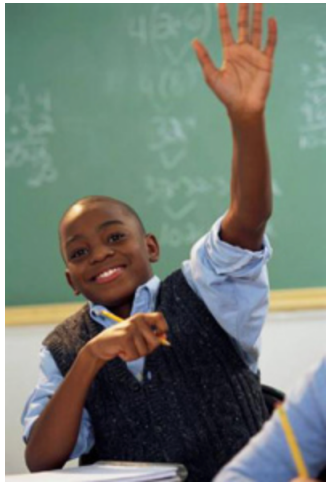
© 1940 Walt Disney

Personas

Personas are useful when real users are not available or are too numerous to interview them all.

- Important class of user, with unique needs and characteristics
- Include enough details (including a name!) to make the persona seem real to the team

Ken
(the keener)



Dudley
(the distracted)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Creativity-based elicitation

Idea: To invent undreamed-of requirements that bring about innovative change and gives competitive advantage.

- Document studies
- Similar companies
- Norms
- Domain analysis
- Requirements taxonomies
- Modelling
- Analysis patterns
- Mockups & prototyping
- Pilot experiments
- Stakeholder analysis
- Questionnaires
- Interviews
- Observation
- Task Demo
- Ask suppliers
- Domain workshop
- Personas
- *Systemic Thinking*
- *Brainstorm* *Creativity-based*
- *Creativity workshop*

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

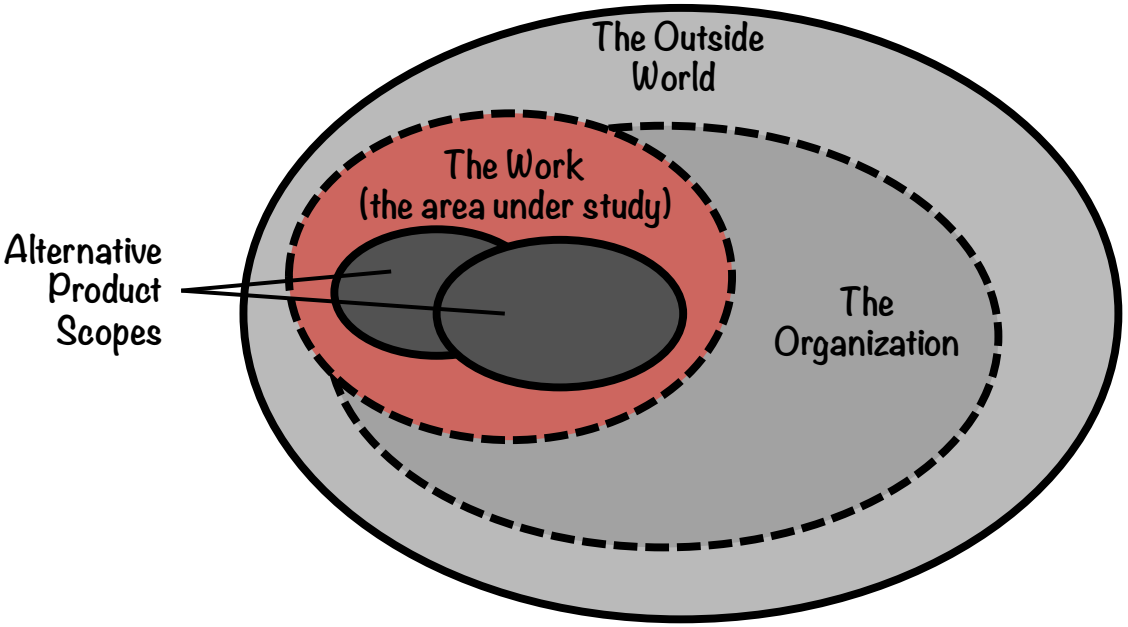
Innovation

"Our job is to give the client, on time and on cost, not what he wants, but what he never dreamed he wanted; and when he gets it, he recognizes it as something that he wanted all the time."

—Denys Lasdun, architect

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Systemic Thinking



Robertson, Robertson, Mastering the Requirements Process, 2012, Figure 3.3.

Thinking systemically about the Work (and not just the future system)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Brainstorming

Brainstorming is a group creativity technique designed to generate a large number of new ideas

Goals:

- Want to hear ideas from everyone, especially unconventional ideas.
- Creativity to be encouraged

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Part I — Idea Generation

- Goal is to generate as many ideas as possible.
  - Quantity, not quality, is goal at this stage
  - Look to combine or vary ideas already suggested
- Scribe writes down all ideas so that all can see them

  e.g., whiteboard, paper taped to wall

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Part II — Assessment

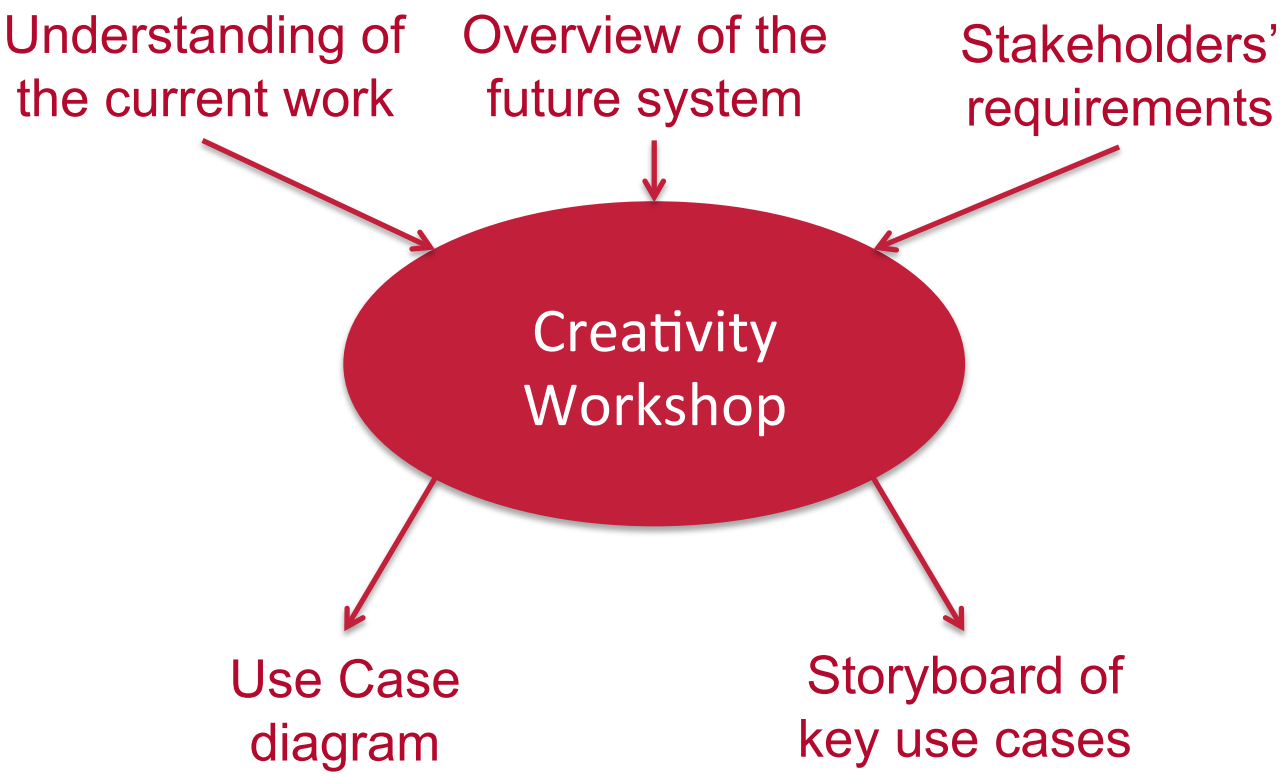
As a separate activity, possibly involving a different set of stakeholders…

- Go over the list and explain ideas more carefully
  - Review, consolidate, combine, clarify, expand.
- Rank ideas and choose winners
- Be careful about time
  - Creative / technical meetings tend to lose focus after 90 min.
  - Take breaks or reconvene later.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

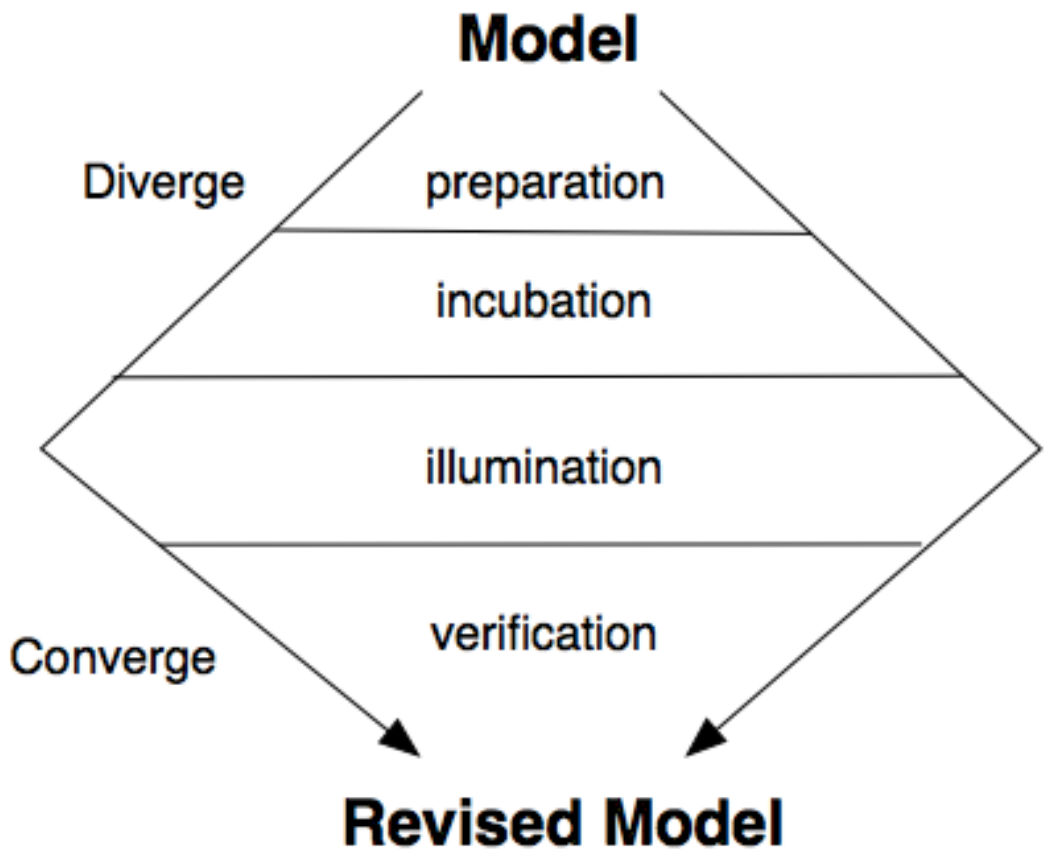
Creativity Workshops

A risk-free space for creating and inventing news ideas – over and above the stakeholders' expressed requirements.



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Creativity Workshops



Adapted from N. Maiden and S. Robertson, "Integrating Creativity into Requirements Processes", International Requirements Engineering Conference, 2005.

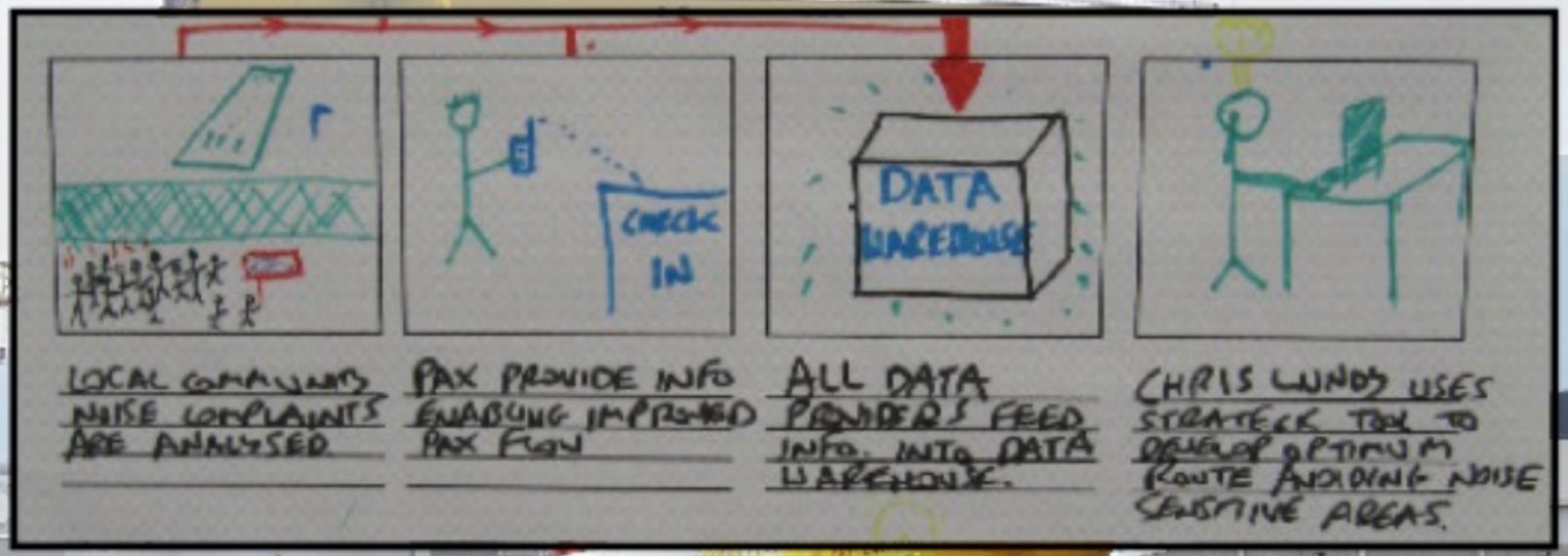
CITY UNIVERSITY LONDON

Centre for Creativity in Professional Practice

A Real-World Example

Removed key constraint: weather variability

Steam catapults; glided approaches; weather-adapted approach routes



.. not what they wanted, but what they never dreamed they wanted..

CITY UNIVERSITY LONDON

Centre for Creativity in Professional Practice

A Second Real-World Example

Specifying concept for new conflict resolution support

- Indian textile expert encouraged incubation about requirements on patterns
- French ATCos highlighted need for aesthetics in generated resolutions

..not what they wanted, but what they never dreamed they wanted..

CITY UNIVERSITY LONDON

Centre for Creativity in Professional Practice

A Third Real-World Example

Rich storyboards to describe concept of operation for multi-sector planning

- Complete system view
- Film screenplay techniques
- Constructed 2 large storyboards for whole system over 4 hours period
- Participants invented semantics of storyboard
- Tactile and flexible
- Ownership important

.. not what they wanted, but what they never dreamed they wanted..

Creativity Workshops

Creativity-based elicitation techniques include

- Open brainstorming
- Analogical reasoning: exploring analogies to a related problem
- Combining ideas
- Exploring new possibilities that result from relaxing identified constraints

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Managing Expectations

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Elicitation Techniques

	Early elicitation	Detailed elicitation	Elicit existing requirements	Elicit new and innovative requirements
Document Analysis				
Questionnaire				
Interview				
Ethnographic Analysis				
Modelling				
Prototyping				
Brainstorming				
Creativity Workshop				

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Elicitation is hard

- It's common to use a physically violent metaphor for elicitation, like extracting a painful tooth or trawling for tuna.
- It is through the acts of *discussing, analyzing, modelling, reviewing, negotiating over, inventing, synthesizing,* and *explicitly documenting* that the various stakeholders come to a mutual understanding and agreement about what the real requirements are (and are not).

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Deliverable #4

- (10%) Updated Use-Case Diagram
- (40%) Process Model (or Activity Diagram)
  - one model per use case (N key cases for a team of size N)
  - cite the source of the requirements for each process/activity
- (50%) Scenarios
  - one scenario per use case (N key cases for a team of size N)
  - use the scenario template provided in the lecture slides
  - include at least 4 notable alternatives and 2 notable exceptions per scenario
  - cite the source of the requirements for each step in the scenario's normal case, alternatives, and exceptions

your models should reflect requirements that are elicited from at least two (2) concrete stakeholders using at least four (4) different elicitation techniques

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

designing the user experience

upa usability professionals' association
www.upassoc.org

Disdaimer: Digital or hard copies of this file may be distributed for personal, internal corporate, or classroom use provided that copies are not made or distributed for profit or commercial advantage. Copies must bear this notice and the UPA logo. Any other use requires specific permission from UPA.

© 2000 Usability Professionals' Association

SE463
Software Requirements Specification & Analysis

Functional Requirements

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Functional Requirements

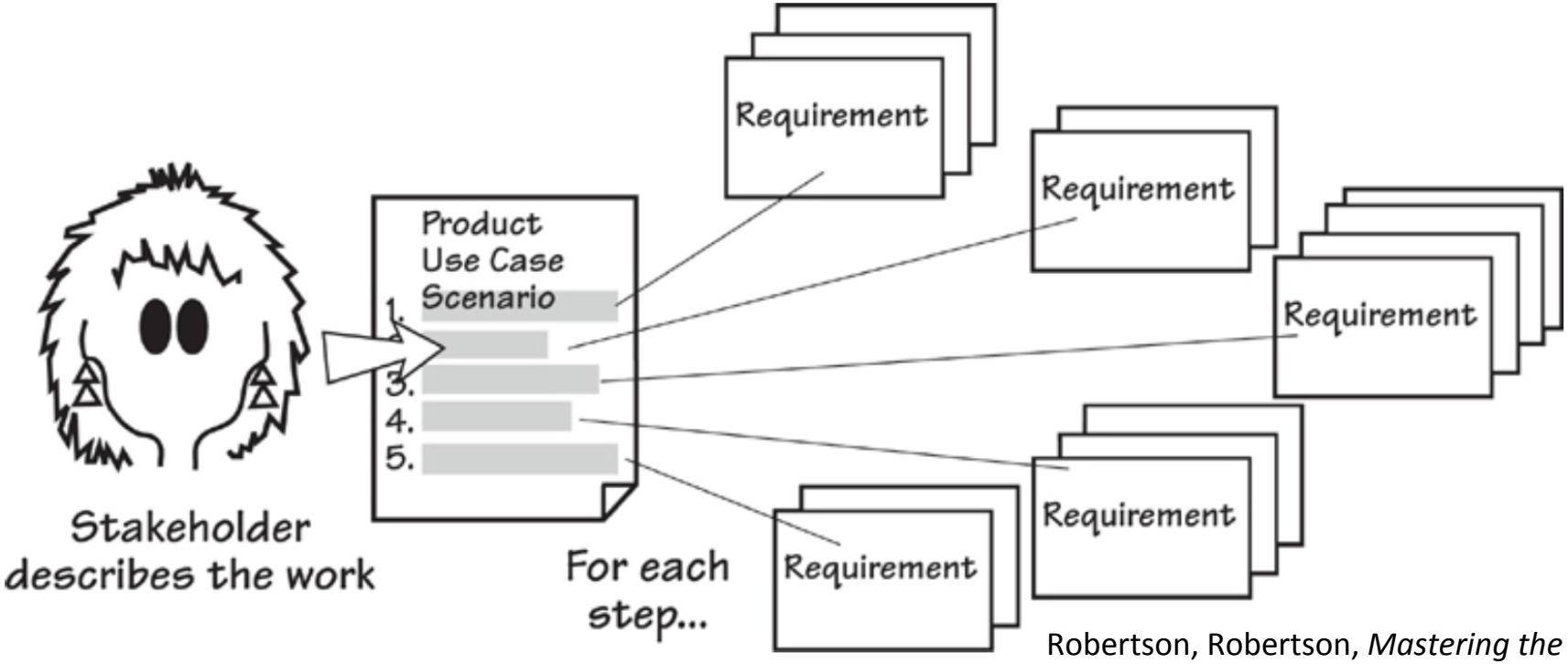
Functional requirements describe what the product has to do to support and enable the Work.

They should be, as far as possible, independent of the technology used by the eventual product.

- functional requirements
- EARS (Easy Approach to Requirements Syntax)
- user stories
- graphical function models

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Atomic Requirements



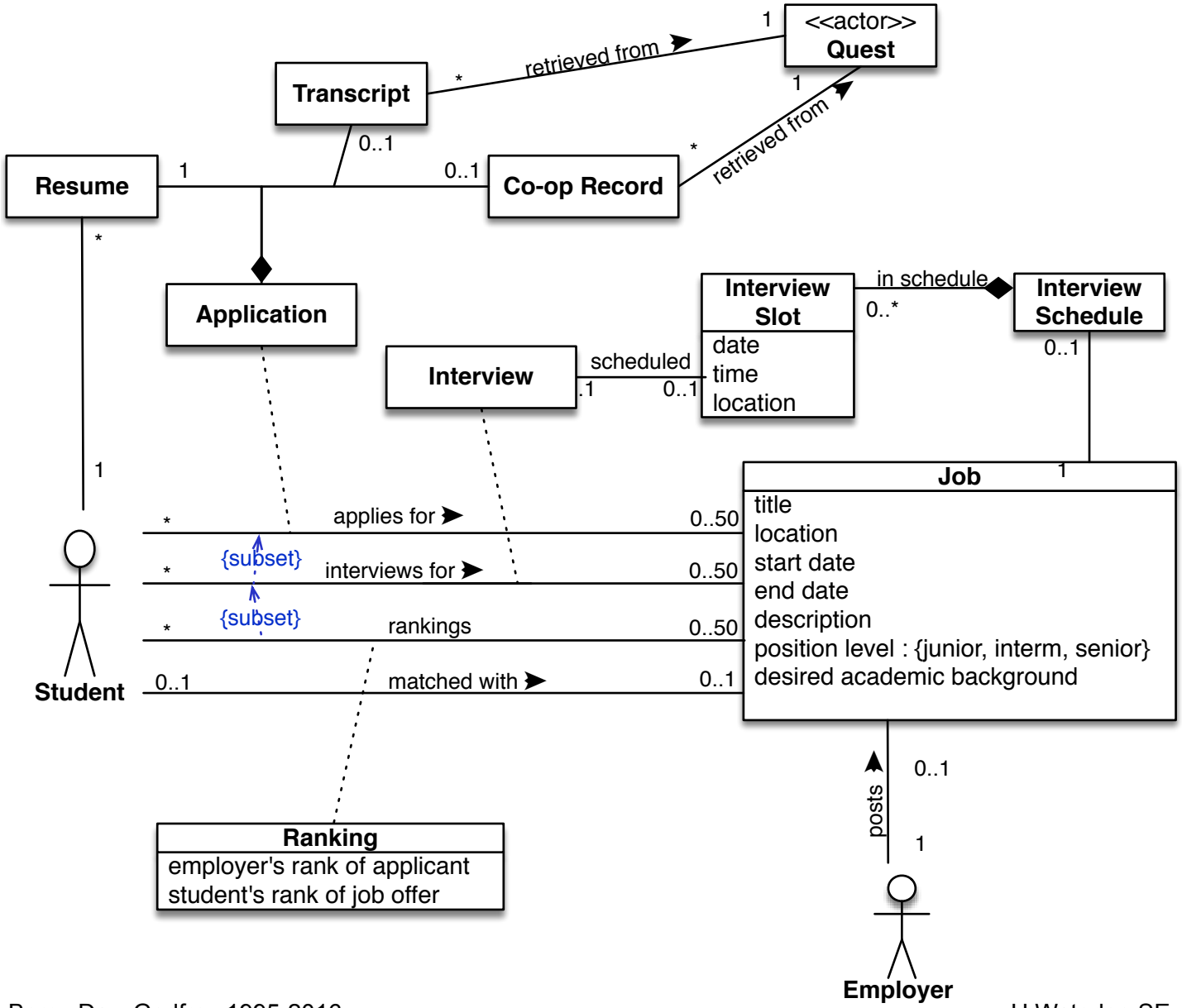
Robertson, Robertson, *Mastering the Requirements Process*, 2012, Figure 10.2

Atomic (functional) requirements are derived from scenarios or activity diagrams

- What the product has to do to achieve each step/activity
- Expressed using vocabulary from the domain model

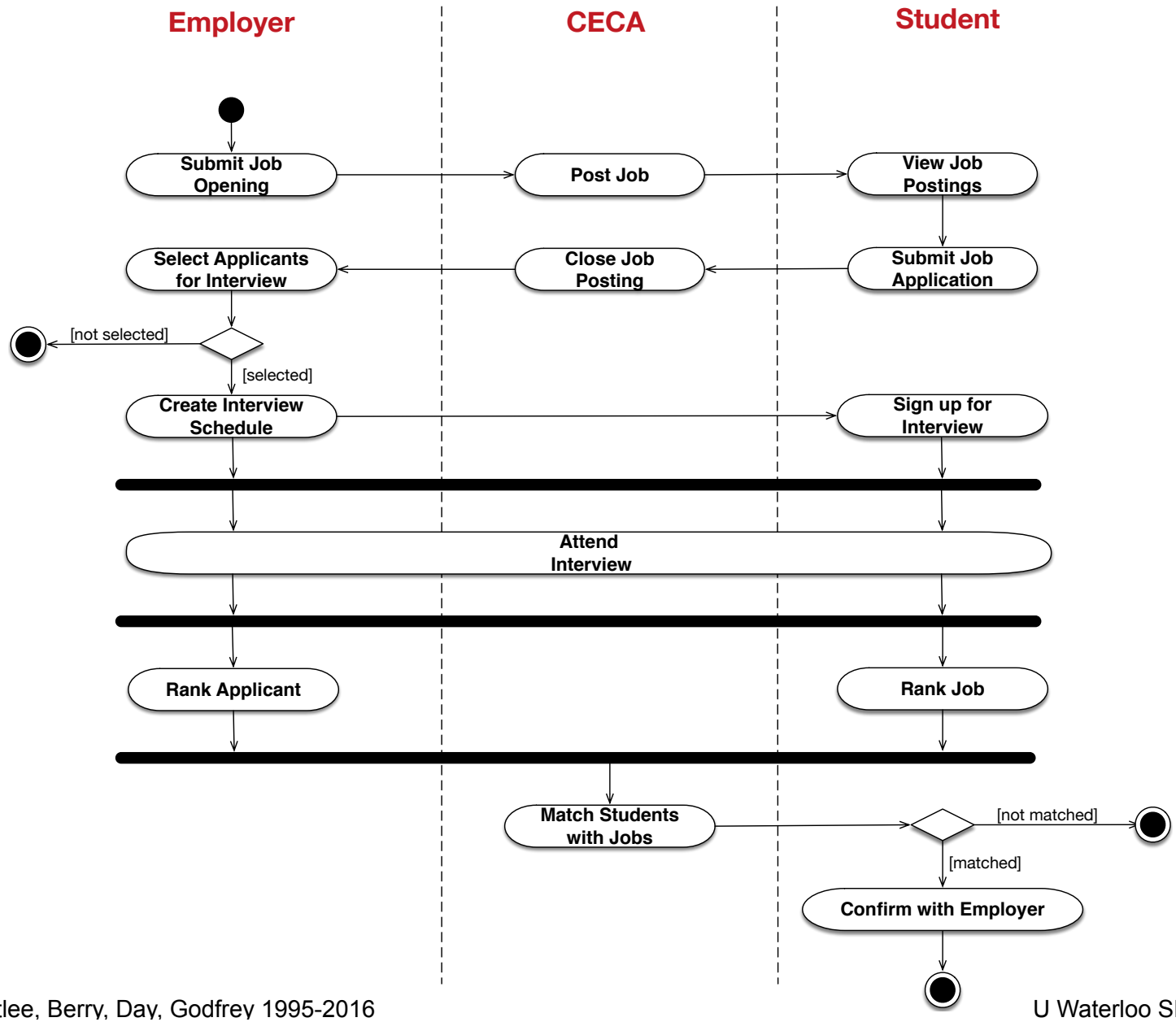
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

JobMine Domain Model



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

JobMine Activity Diagram



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Level of Detail or Granularity

Atomic requirements are written as a single sentence with a single verb.

Product will require a ranking for each job where the student user has been ranked by the employer

Do *not* use a mixture of modal verbs (shall, must, might, could) to indicate the priority of a requirement.

- Use a single modal verb consistently, and indicate priority as an attribute of the requirement.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Alternatives and Exceptions

Include atomic (functional) requirements that are derived from scenario alternatives and exceptions.

These requirements are conditional on the event / condition that leads to the alternative or exception.

The product will display the set of available interview slots and allows the student user to select one of them.

- *If the student user neglects to select an interview time, then two days before the interview date the product will automatically schedule the student into one of the remaining interview slots.*

The product will confirm receipt of a student user's submitted job application.

- *If submission of a job application fails, the product will warn the user that the submission failed.*

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Rationale

Each atomic requirement should be accompanied by a rationale that explains why the requirement exists.

If there are multiple matches whose sums of employer and student rankings are all the lowest sum, the product will select among these matches at random

Rationale: To ensure that job matches are not biased toward employer or student rankings.

The product will record the start and end times of a student user's session when viewing job applications.

Rationale: CECA wants to know which users use the product the most.

Rationale: Viewing job applications should take 90% of students less than 6 hours a week.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Requirements Template

Requirement #: 75 Requirement Type: 9 Event/BUC/PUC #: 7, 9

Description: The product shall record all the roads that have been treated

Rationale: To be able to schedule untreated roads and highlight potential

Originator: Arnold Snow - Chief Engineer

Fit Criterion: The recorded treated roads shall agree with the drivers' road treatment logs and shall be up to date within 30 minutes of the completion of the road's treatment

Customer Satisfaction: 3 Customer Dissatisfaction: 5

Dependencies: All requirements using road and scheduling data Conflicts: 105

Supporting Materials: Work context diagram, terms definitions in section 5

History: Created February 29,2010

Volere

Copyright © Atlantic Systems Guild

Robertson, Robertson, *Mastering the Requirements Process*, 2012, Appendix A, Figure A.10

EARS Template

Easy Approach to Requirements Syntax (EARS)

A template for structuring the syntax of the one-sentence description of an atomic requirement

- acknowledges that the vast majority of requirements will be expressed in natural language
- promotes a gently constrained use of natural language

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

EARS Generic Syntax

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

<optional precondition> <optional trigger> the <system name> shall <system response>

The generic requirement syntax is specialized into five types of requirement

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

EARS Ubiquitous Requirements

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

A ubiquitous requirement is an invariant behaviour of the system (that should always hold).

The <system name> shall <system response>

Example:

"The control system shall prevent engine overspeed."

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

EARS Event-Driven Requirements

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

An event-driven requirement applies on when a triggering event is detected at the system boundary.

WHEN <trigger> the <system name> shall <system response>

Example:

"When commanded by the aircraft, the control system shall switch on continuous ignition"

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

EARS State-Driven Requirements

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

An state-driven requirement applies as long as the system is in a particular state, or mode of operation.

WHILE <in a specific state> the <system name> shall <system response>

Example:

"While the aircraft is in-flight, the control system shall maintain engine fuel flow above XXlbs/sec"

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

EARS Option Requirements

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

An option requirement when system behaviour is dependent on the presence of a particular feature.

WHERE <feature is included> the <system name> shall <system response>

Example:

"Where the control system includes an overspeed protection function, the control system shall test the availability of the overspeed protection function prior to aircraft dispatch."

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

EARS Unwanted Events

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

Special syntax distinguishes required system responses to unwanted events (e.g., failures, disturbances, deviations from desired user behaviour or any unexpected behaviour of interacting systems).

IF <trigger>, THEN the <system name> shall <system response>

Example:

"If the computed airspeed fault flag is set, then the control system shall use modelled airspeed."

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

EARS Combination Requirements

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

Examples:

"While the aircraft is on-ground, when reverse thrust is commanded, the control system shall enable deployment of the thrust reverser".

"When selecting idle setting, if aircraft data is unavailable, then the control system shall select Approach Idle"

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Real Examples

Rolls Royce Control Systems	Re-expression in EARS
It must be demonstrated that, when a Fault or Failure results in a change from one Control Mode to another, or from one channel to another, or from the Primary System to the Back-Up System, the change occurs so that the Engine does not exceed any of its operating limitations.	When the Engine Control System changes operational mode, the Engine Control System shall maintain the engine within the approved operation limits.
Single Failures leading to loss, interruption or corruption of Aircraft-Supplied Data must not result in a Hazardous Engine Effect for any Engine.	If a single Failure leads to deficient Aircraft-Supplied Data, then the Engine Control System shall not cause a Hazardous Engine Effect.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

User Stories

User stories provide a light-weight approach to managing requirements:

- Short statement of new functionality or feature
- Written from the point of view of the user
- Its details are fleshed out just in time before the story is placed in the sprint.

Three C's of User Stories

- **Card** - stories are traditionally written on note cards, using a structured syntax:

  As a <role>, I want <something>, so that <benefit is achieved>

  As a vacation traveller, I want to see photos of hotels, so that I can get a sense of the quality of the hotel

  As a user, I want to cancel a reservation, so that my credit card is not charged

- **Conversation** – discussions with the product owner reveal details of the requirements

- **Confirmations** – acceptance criteria for objectively determining whether an implementation meets the requirements.

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Details as conditions of satisfaction

As a user, I can cancel a reservation.

- The product owner's conditions of satisfaction can be added to a story
- These are essentially tests

- ☐ Verify that a premium member can cancel the same day without a fee.
- ☐ Verify that a non-premium member is charged 10% for a same-day cancellation.
- ☐ Verify that an email confirmation is sent.
- ☐ Verify that the hotel is notified of any cancellation.

© Copyright Mountain Goat Software

Details added in smaller sub-stories

As a user, I can cancel a reservation.

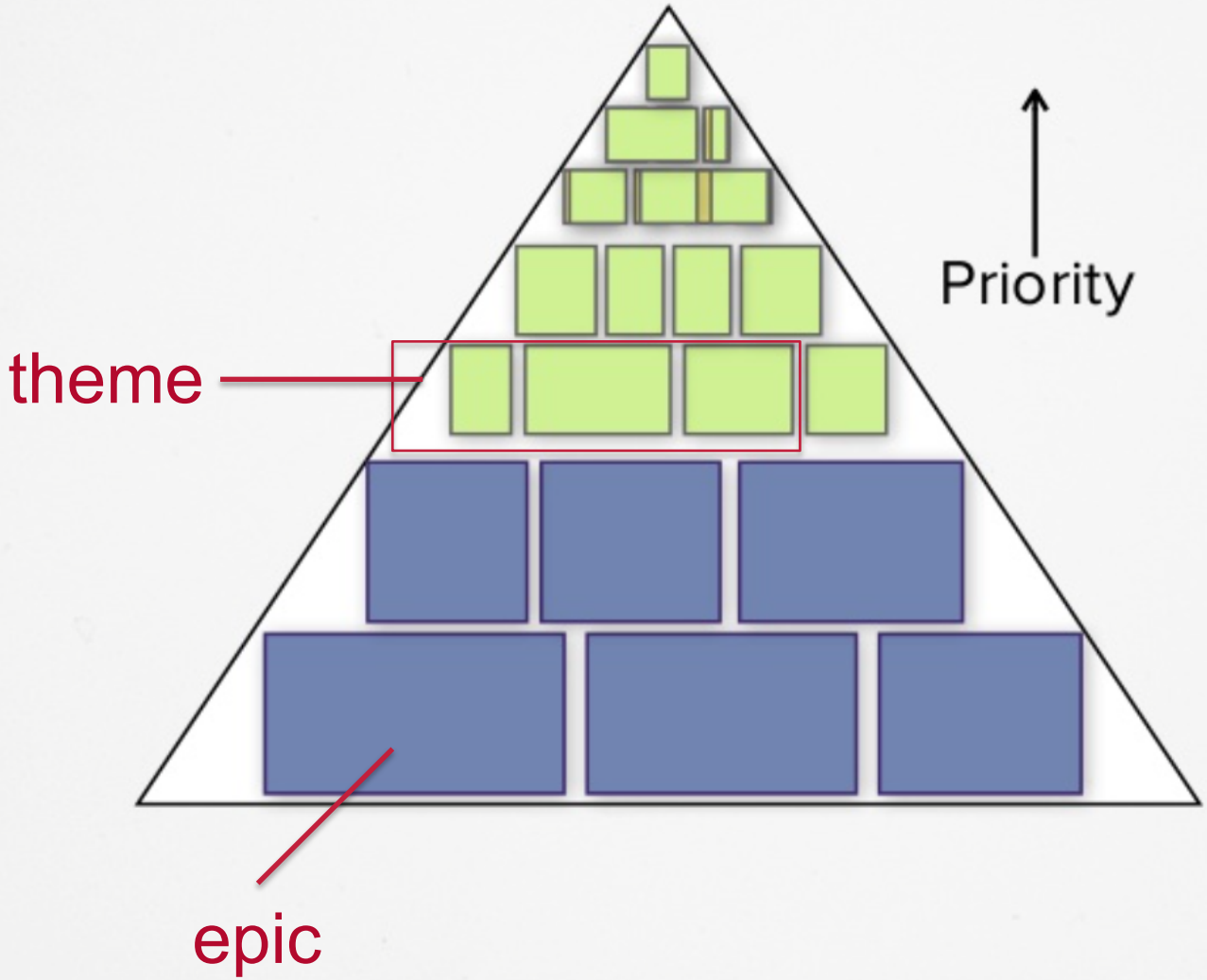
As a premium site member, I can cancel a reservation up to the last minute.

As a non-premium member, I can cancel up to 24 hours in advance.

As a site visitor, I am emailed a confirmation of any cancelled reservation.

© Copyright Mountain Goat Software

The product backlog iceberg



© Copyright Mountain Goat Software

Why User Stories?

- Easy for stakeholders to understand, and to remember.
- Shift the focus from written documentation to discussion.
- Encourage iterative development, with stories being appropriate sized increments for planning
- Delay the elicitation of requirements details until just before development.
- Support participatory elicitation

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

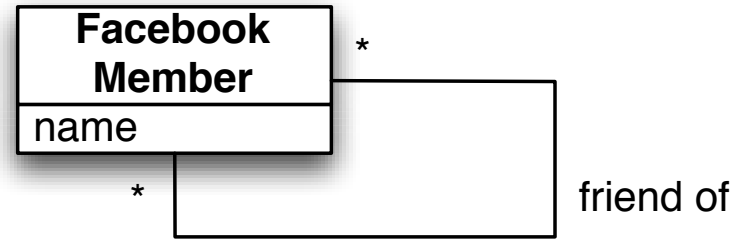
Deliverable #5

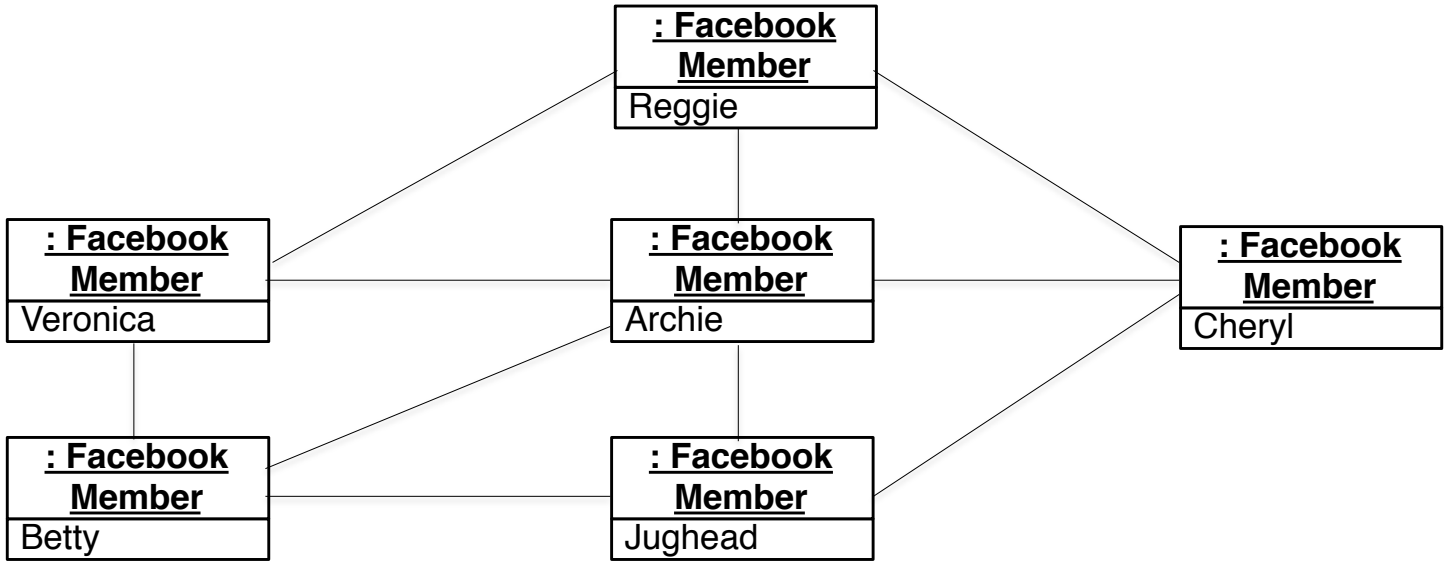
- EARS requirements or User Stories
  - For all atomic requirements needed to implement the steps of all of your scenarios in Deliverable #4
  - Including the atomic requirements needed to implement the steps in your scenarios' alternative and exception paths
  - Cite the source of each atomic requirement and the elicitation technique that was used
  - Your atomic requirements should
    - reflect the needs of at least three (3) stakeholders
    - elicited from at least two (2) concrete stakeholders
    - using at least four (4) elicitation techniques

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Function as a Change to the World State

The domain model represents the set of possible states of the world (called world states)

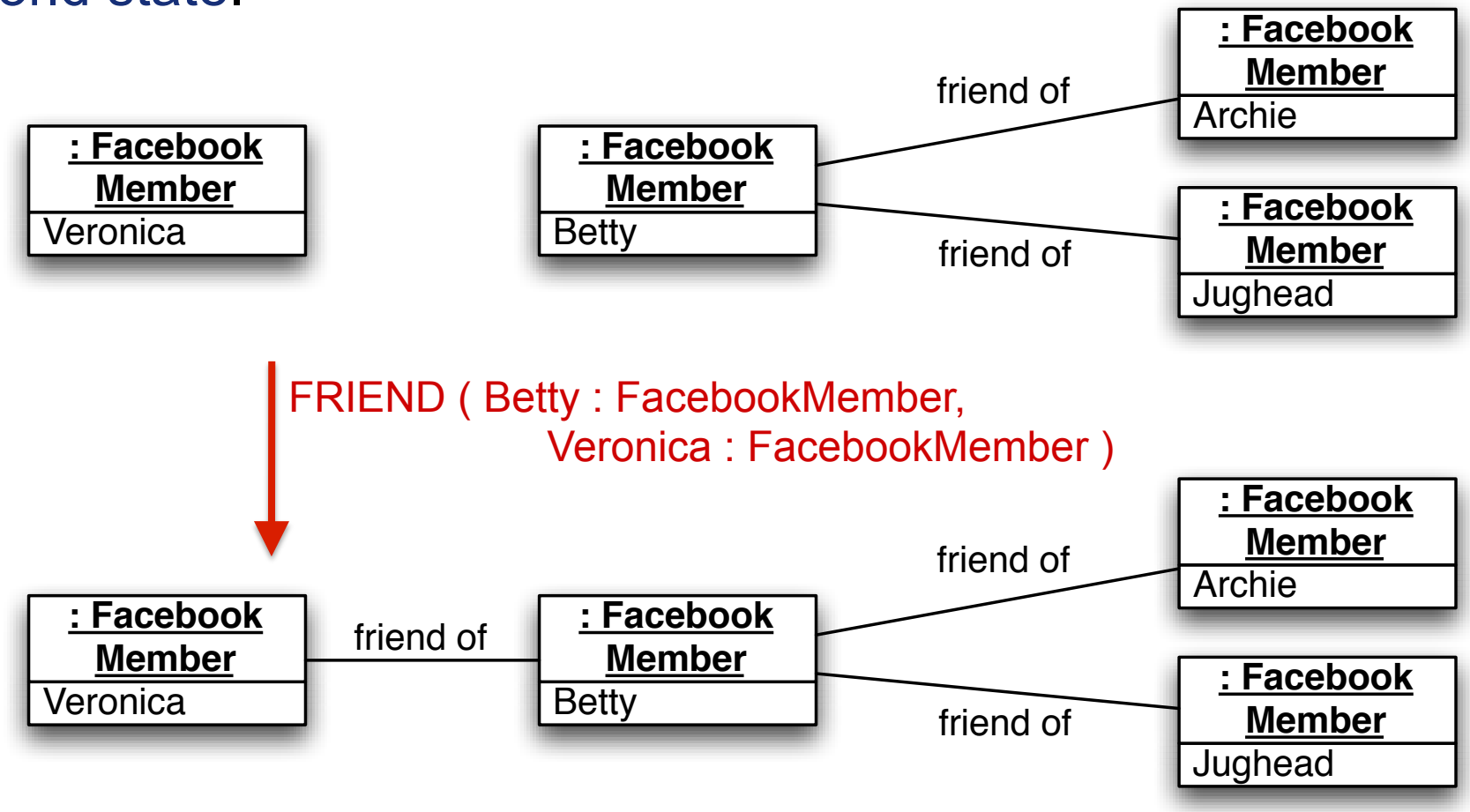




Copyright © Atlee, Berry, Day, Godfrey 1995-2016

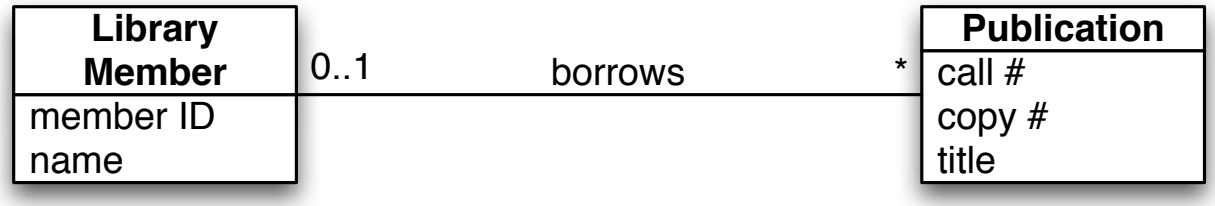
Graphical Function Model

Graphical Function Model – expresses a system-level function as a rule that matches and replaces patterns in the world state.



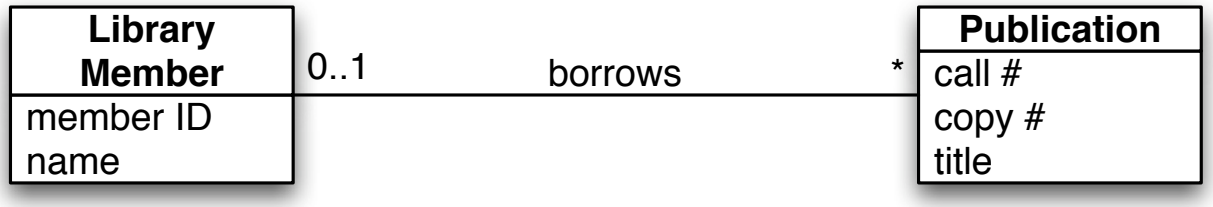
Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Graphical Functional Model

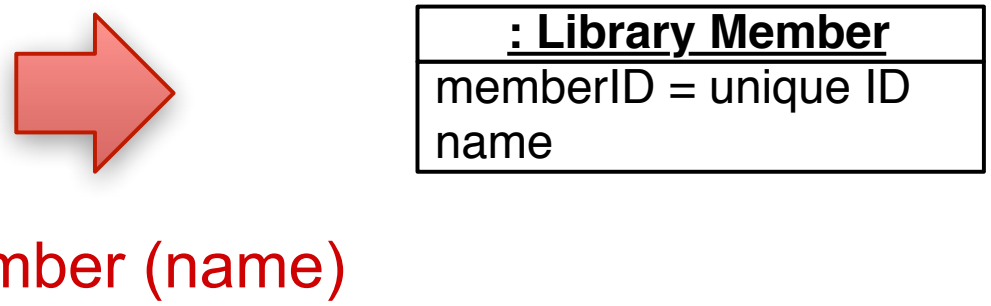


Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Another Example

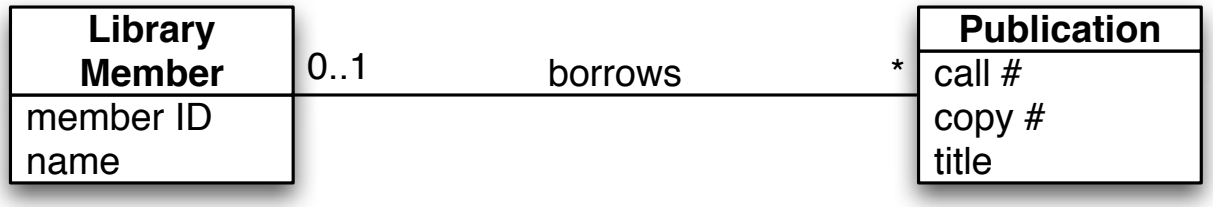


AddMember (name)



Copyright © Atlee, Berry, Day, Godfrey 1995-2016

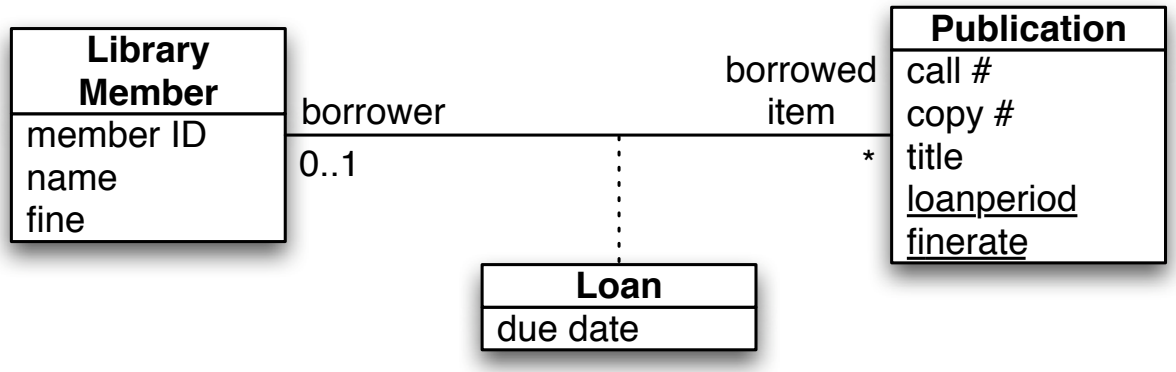
Another Example



DeleteMember(memberID)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Another Example



BorrowItem (m:LibraryMember; p:Publication; today)

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Express atomic requirements as system-level functions.

- functional requirements
- EARS (Easy Approach to Requirements Syntax)
- user stories
- graphical function models

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

SE463
Software Requirements Specification & Analysis

Conflict Resolution

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Multiple Viewpoints

Different stakeholders may have conflicting ideas about what the software should do.

Example: Students Records Information System

- Who are the stakeholders?
- What information might each stakeholder want?

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Conflict Analysis

- Data conflict: multiple conflicting understandings of an issue

Example:

- Student information system should track attendance
- There is no room to record attendance on a transcript

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Conflict Analysis

- Data conflict: multiple conflicting understandings of an issue
- Interest conflict: stakeholders have subjectively or objectively different goals or interests

Example:

- Province demands that student information system track attendance, for accounting purposes
- Students insist that attendance not be recorded, to protect their privacy

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Conflict Analysis

- Data conflict: multiple conflicting understandings of an issue
- Interest conflict: stakeholders have subjectively or objectively different goals or interests
- Value conflict: stakeholders express different preferences

Example:

- Engineering Faculty wants to rank students in courses
- Students think that rankings are too fine a distinction between students' performances

Copyright © Atlee, Berry, Day, Godfrey 1995-2016

Summary

Conflict Management

- Types of conflict
- Group resolution strategies
- Individual resolution strategies
- Quality of resolution vs. Time to decision
- Importance of Goal vs. Relationship

Copyright © Atlee, Berry, Day, Godfrey 1995-2016