

CMPE 276 Software Engineering

Lecture 2 Software Process

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

1

Course Outline

- | | |
|---------------------------|---------------------------|
| • 9/19 Overview | • 10/24 Midterm |
| • 9/26 Reqmnts and spec | • 10/29 Sw model checking |
| • 10/1 UML | • 10/31 Testing/verif |
| • 10/3 Design patterns | • 11/5 Dynamic Analysis |
| • 10/8 Config management | • 11/7 Static Analysis |
| • 10/10 Testing | • 11/12 Embedded Sw |
| • 10/15 Project present. | • 11/14 Case study |
| • 10/17 Testing: Verisoft | • 11/19 Formal specs |
| • 10/22 Sw model checking | • 11/21 Project present. |
| • 10/22 Sw model checking | • 11/26 Interface specs |
- Marielle Stoelingsa
 Jim Whitehead
 Guest lecturer

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

2

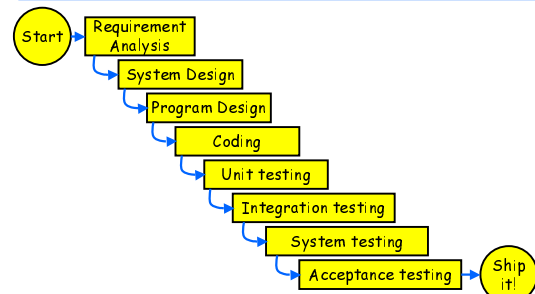
Software process

- **Process**: the activity of building a product (software program, house, IC, TV set, ...)
- Where do we start? What is the sequence of activities?
 - Most projects go through common phases (requirement gathering, specification, implementation, testing...)
 - Lots of empirical and accumulated knowledge.
- How do we estimate the effort?
 - Cost/time estimation
 - Ensuring that the resources are in place (personnel, facilities, tools, ...)

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

3

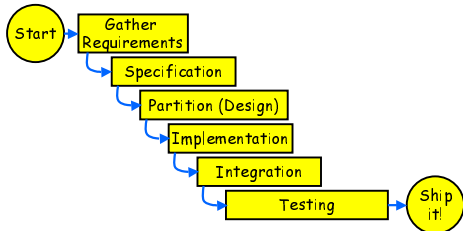
The Waterfall Model (officially)



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

4

The Waterfall Model ("builder's view")

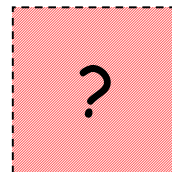


[Aiken, cs169] Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

5

Gather Requirements

- What should the system do?
 - Talk to customer, user (if available), marketing
 - Note that they don't always know what they want!
 - Wish list of features
 - Rough cost/time estimates

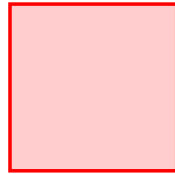


Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

6

Specification

- Write a specification of what the system should do
 - and of what it requires in order to work
- Specification must be complete
 - Describe behavior in all circumstances
 - Can be large
 - How are details filled in?

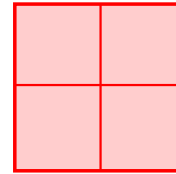


Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

7

Partition (Design)

- Decide system architecture
- Divide the design into manageable pieces
 - According to functionality
 - Split along stability lines
- Specify the interfaces between modules

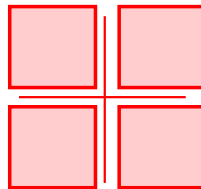


Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

8

Partition (Design)

- Decide system architecture
- Divide the design into manageable pieces
 - According to functionality
 - Split along stability lines
- Specify the interfaces between modules

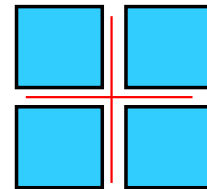


Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

9

Implementation

- Make a plan, guided by:
 - Priority of features
 - Ability to test the design
 - Critical components first
- Implement the components
 - Or re-use components, if possible
- Test the components
 - May require scaffolding
 - Order of implementation is important

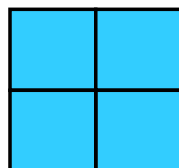


Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

10

Integration

- Put everything together
- Test it
 - For functionality
 - According to interfaces
- What if...
 - Some components are late (can we integrate/test the others?)
 - Errors in interfaces (more expensive to fix)

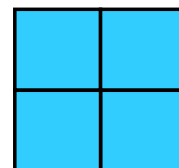


Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

11

System and Acceptance Testing

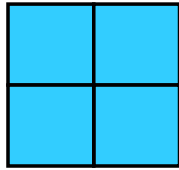
- System testing
 - Does the system work according to the specifications?
 - Mayor effort
 - When to ship? (0 bugs may be unattainable)
- Acceptance testing
 - Does the customer like the system?
 - Beta-releases



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

12

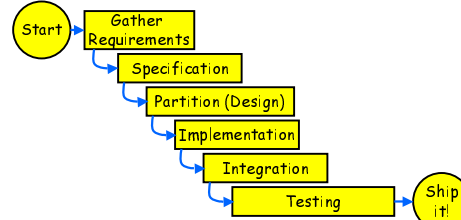
Ship it!



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 13

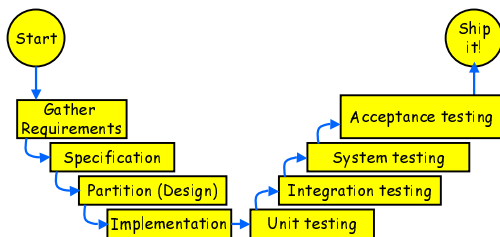
The Waterfall model

- Top-down specifications
- Bottom-up implementation and testing



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 14

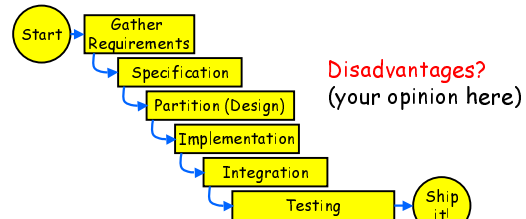
The V Model (relating tests to specs)



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 15

The Waterfall model

- Top-down specifications
- Bottom-up implementation and testing



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 16

Waterfall Model - Disadvantages

- Little feedback from later phases on early ones.
 - If the specification is wrong, and this is discovered during acceptance testing, a lot of work needs to be re~done!
- Long time before anything useful is obtained
 - Difficult to get user input
 - It is much easier for the user to react to a working system, and revise the requirements, than to give a consistent and complete set of requirements at the beginning.

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 17

Waterfall Model - Shortcomings

- Difficult to write all-encompassing specifications in the absence of an implementation.
- Little usable outcome if the project has to be shut/reoriented in midcourse.
- If delivery time is long:
 - User requirements are likely to change, causing instability (and more bugs)
 - Competing products coming out (another cause for requirement change)

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 18

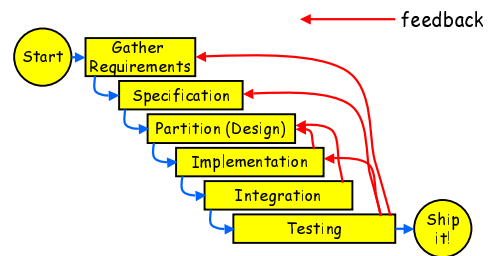
Waterfall Model - Good Points

- Emphasis on specifications, and on the verification that the specifications are met.

How often is it used in practice?

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 19

The Waterfall Model in Practice



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 20

Rapid Prototyping

- Get a rough version of the requirements
- Write a quick prototype and show it to the user
 - To demonstrate functionality
 - Can be a mock-up only (not distributed, ...)
- Get refined requirements
- Proceed as in the Waterfall model
 - Throw away prototype
 - Design and implement new system

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 21

Rapid Prototyping

- Get a rough version of the requirements
- Write a quick prototype and show it to the user
 - To demonstrate functionality
 - Can be a mock-up only (not distributed, ...)
- Get refined requirements
- Proceed as in the Waterfall model
 - Throw away prototype
 - Design and implement new system

Disadvantages?

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 22

Rapid Prototyping - Disadvantages

- Time invested in the prototype
 - Tendency to over-do it to please the customer
 - Demos, demons
- Difficult to throw away the prototype and do a clean full-system design
 - Some code of the prototype tends to make it to the full system
 - and sub-optimal design choices (architectures, algorithms) with it
- Potentially very useful if done correctly
 - Easier to evolve requirements based on a prototype
 - Can be used to experiment with architectures

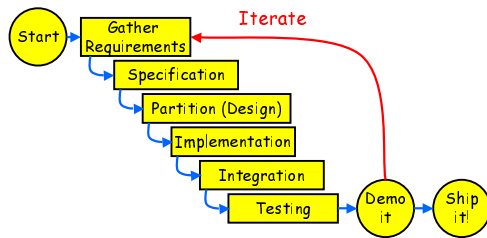
Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 23

Iterative Models

- Evolve the system as a sequence of prototypes: from rough ones, used to evolve requirements, to advanced ones with more and more functionality.

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 24

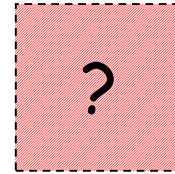
Iterative Models



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 25

Gather Requirements

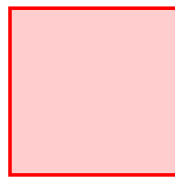
- Get preliminary requirements



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 26

Specification

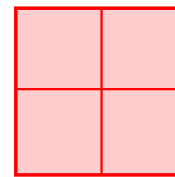
- Write a specification of what the system should do
 - Focus on the main functionality first
 - Explore some "reasobable" assumptions for under-specified requirements.



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 27

Partition (Design)

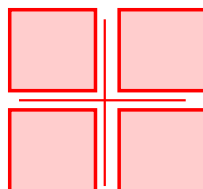
- Decide system architecture
- Divide the design into manageable pieces
 - According to functionality
 - Split along stability lines
- Specify the interfaces between modules



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 28

Partition (Design)

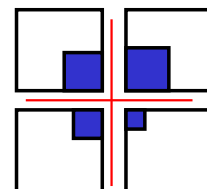
- Decide system architecture
- Divide the design into manageable pieces
 - According to functionality
 - Split along stability lines
- Specify the interfaces between modules



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 29

Implementation

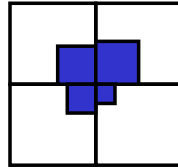
- Make a plan, guided by:
 - Priority of features
 - Ability to test the design
 - Critical components first
- Implement the components
 - Do just the bare-bone functionality first
- Aim: getting quickly a complete system build, that can do very little.



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 30

Integration and Testing

- Put everything together
- Test it
 - Are the interfaces well chosen?
 - Test all the functionality present
- If functionality is substantial, can even be shipped.

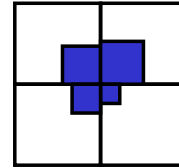


Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

31

Get Feedback

- Get feedback from user and/or customer
- Use the feedback to revise the requirements, and hence the specifications

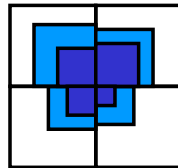


Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

32

Revise Requirements and Increase Functionality

- Revise the prototype (fix incorrect requirements)
- Design and implement more functionality.

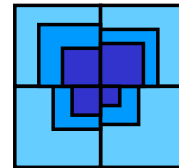


Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

33

Revise Requirements and Increase Functionality

- Revise the prototype (fix incorrect requirements)
- Design and implement more functionality.
- Test it.
- Continue until sufficient functionality is reached to ship (may ship versions with increasing functionality).



Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

34

Rapid Prototyping

Advantages:

- Much easier to get correct requirements
 - Hopefully, they won't change as much after the first few prototypes
 - Feedback on experimental features
- Experiment with architectural choices early.
- Refine design (algorithms, ...) only where needed.
- Even if development is stopped early, some functionality may be available.
- Quantifiable (75% of functionality implemented)

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

35

Rapid Prototyping

Disadvantages:

- Much easier to get correct requirements
 - Hopefully, they won't change as much after the first few prototypes
 - Feedback on experimental features
- Experiment with architectural choices early.
- Refine design (algorithms, ...) only where needed.
- Even if development is stopped early, some functionality may be available.
- Quantifiable (75% of functionality implemented)

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

36

In practice

- **Waterfall model:**
 - Systems that must work correctly from the beginning
 - Systems that are difficult to test, or demo
 - Spacecrafts, air traffic control, banking systems, ...
- **Iterative model:**
 - Consumer software (office productivity, etc)
 - Can be easily demoed, previewed
 - When requirements are fuzzy...

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

37

Cost/Time/Performance Estimation

- Early on, customers typically want:
 - Cost estimate
 - Time estimate
 - Performance estimate
- One of the hardest problems!
- And one that is most often gotten wrong:
 - DMV registration: 87-93, 6.5x cost estimate, 2x time estimate
 - Denver airport baggage system: after one year slippage, cost of \$1.1 millions a day
 - FAA AAS (Advanced Automation System): 5 years late, \$1 billion over budget

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

38

The "Lines of Code" Fad

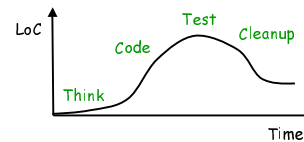
- Is it really a good criterion??
 - Many languages require different amount of lines to do the same
 - Non robust:
 - loop unrolling
 - style of variable declaration, coding
 - COBOL is a high productivity language! (steer away from python or ML, if you want high productivity)
 - It reminds me of alchemists, that were trying to measure the similarity of elements using completely inappropriate measurements.
 - Still, what to use otherwise? Procedure interaction? Flow graph analysis? ...

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

39

The "Lines of Code" Fad

- In my personal experience:



- What about you?

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

40

Lines of Code - The Light Side

- From Alexander Dumas, "The Three Mosqueteers":
 - Qui? demanda Son Eminence
 - Elle et lui.
 - La reine et le duc? s'ecria Richelieu.
 - Oui.
 - Et ou cela?
 - Au Louvre.
 - Vous en etes sur?
 - Parfaitement sur.
 - Qui vous l'a dit?
 - ...

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

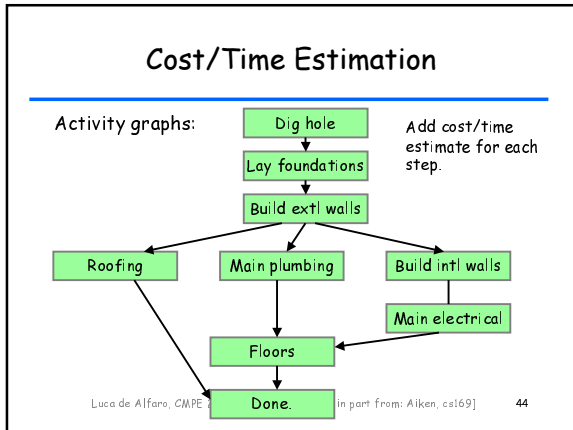
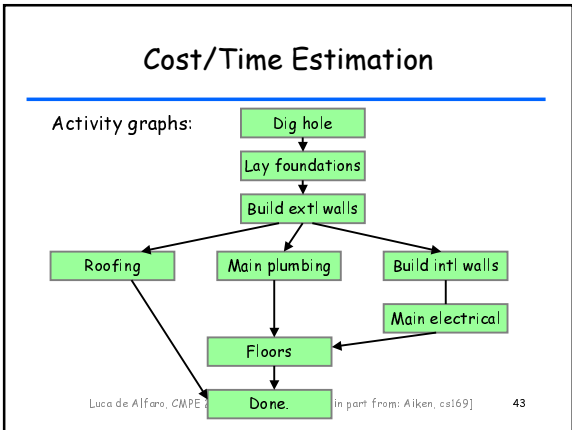
41

Cost/Time/Performance Estimation

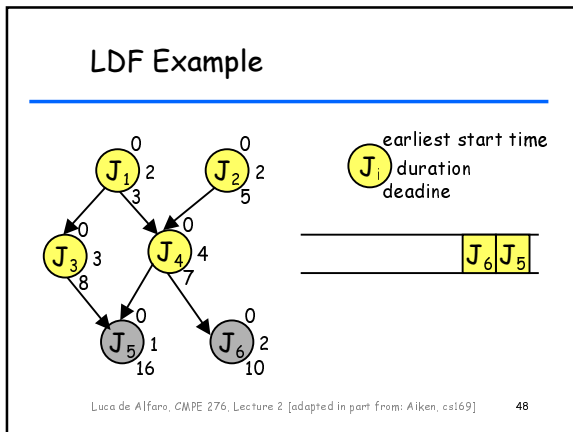
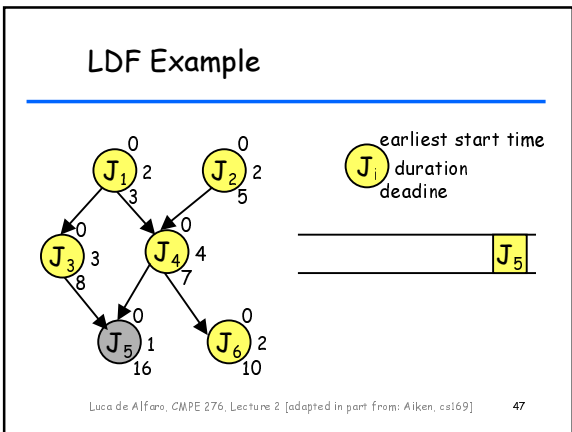
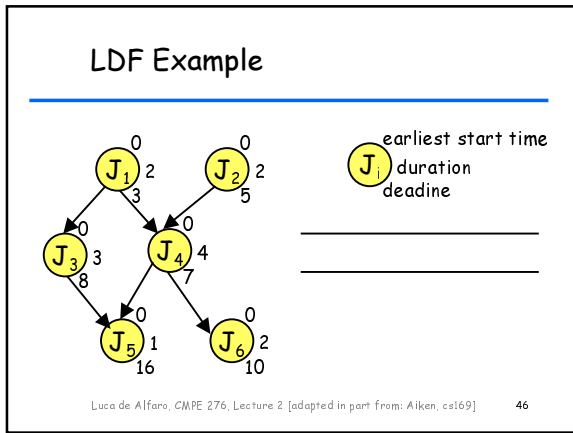
- How to do estimates?
- Empirical models:
 - Measuring parameters
 - How much real-time code?
 - How much distributed processing?
 - How big a user interface?
 - ...
 - Training a model based on previous experiences
 - Using trained model to do forecasts.

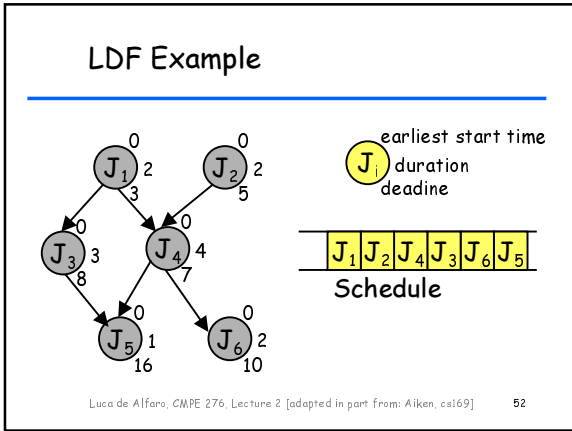
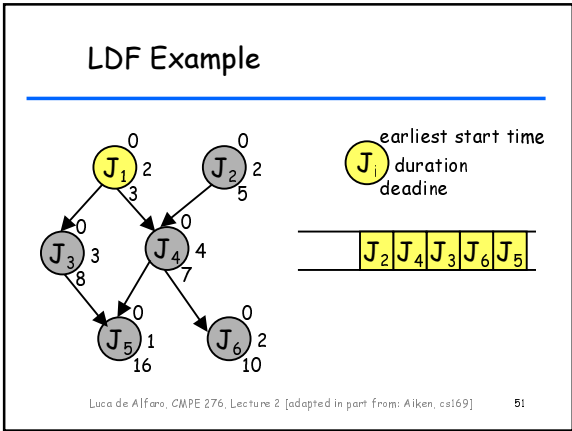
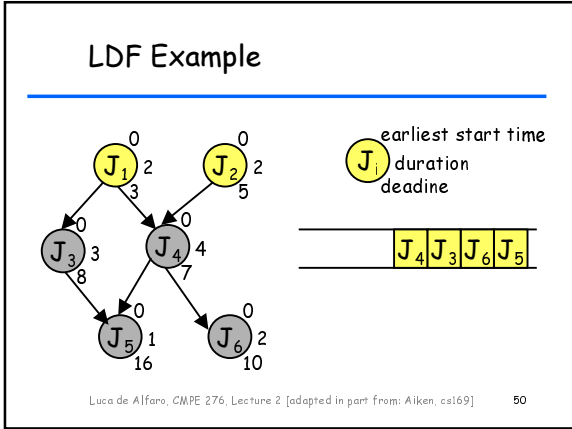
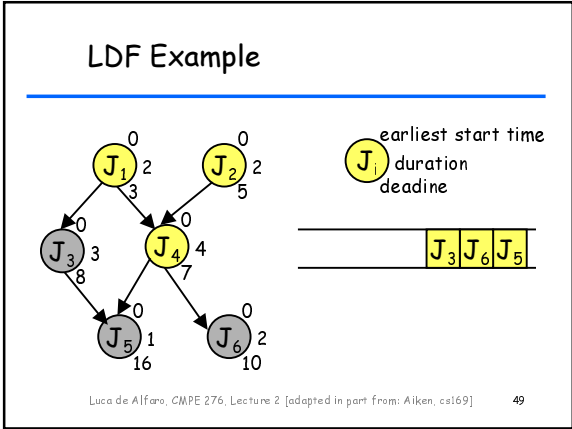
Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

42



- ### Scheduling activity graphs: LDF (Latest Deadline First)
- Build the schedule from last to first, by recursively:
 - Let Q be the subset of activities all whose descendants have been scheduled.
 - Pick the activity p from Q that has the latest deadline.
 - Add p to the front of the schedule.
 - Should be really called LDL (Latest deadline last).
 - Optimal wrt. *maximum lateness*:
 $end_time - deadline$
- Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 45





LDF: Proof of Optimality

S A J_a B J_b

S A B J_b J_a

J_a should be last according to LDF

$L_{max} = \max \{L_A, L_B, L_a, L_b\}$
 L_A unchanged
 $L'_B \leq L_B$ as B starts earlier
 $L'_b \leq L_b$ as J_b starts earlier
 $L'_a = f'_a - d_a \leq f_b - d_b$ (as $d_a > d_b$) = L_b
 So $L'_{max} \leq L_{max}$.
 The argument is concluded by induction.

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 53

- ### Main Risk Factors [Boehm]
- Personnel shortfalls
 - Unrealistic schedules and budgets
 - Developing the wrong functionality
 - Developing the wrong user interface
 - Gold plating
 - Continuing stream of requirement changes
 - Shortfalls in externally performed tasks
 - Shortfalls in externally furnished components
 - Real-time performance shortfalls
- Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169] 54

Main Risk Factors [Boehm][LdA]

- Personnel shortfalls
- Unrealistic schedules and budgets
- Developing the wrong functionality
- Developing the wrong user interface
- Gold plating
- Continuing stream of requirement changes
- Shortfalls in externally performed tasks
- Shortfalls in externally furnished components
- Real-time performance shortfalls
- **Building something the customer doesn't want!!**

Luca de Alfaro, CMPE 276, Lecture 2 [adapted in part from: Aiken, cs169]

55