Iterative Process

CSE 3311 & 5324
Christoph Csallner
University of Texas at Arlington (UTA)

Other source: Federico Biancuzzi and Shane Warden:
Masterminds of Programming: Conversations with the Creators of Major Programming Languages
O'Reilly Media, 2009

Review: Main SE Tasks

What is the problem to be solved?

- Describe domain
- Describe requirements
- Review

How should we solve the problem?

- High-level: architecture
- Low-level: design
- Review

Actually solve the problem

- Write code
- Review, analyze, test, demo

Classic Process (Waterfall)

What is the problem to be solved?

- Describe domain
- Describe requirements
- Review – if okay move to next step

How should we solve the problem?

- High-level: architecture
- Low-level: design
- Review – if okay move to next step

Actually solve the problem

- Write code
- Review, analyze, test, demo – if okay, done

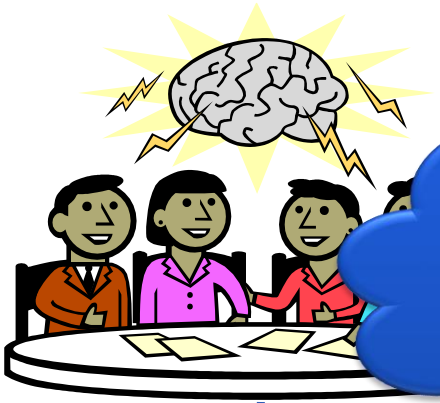
Software Characteristics

- Software is custom built (vs. assembly line manuf.)

- Software is subjected to continual changes
  - Easy to make a change—but hard to make it correctly
  - Changes introduce bugs and deteriorate system structure

- Software processes are (still) not well understood
  - Software dev. processes get in & out of fashion
  - Expensive to collect hard evidence on process quality
    - Would need controlled experiments w/ comparable teams etc.

Problem in All Software Engineering



Consistent with user/customer wishes?

Users and customers

Consistent in itself?

Requirements: Usually **informal,** allows many code solutions

Large Gap

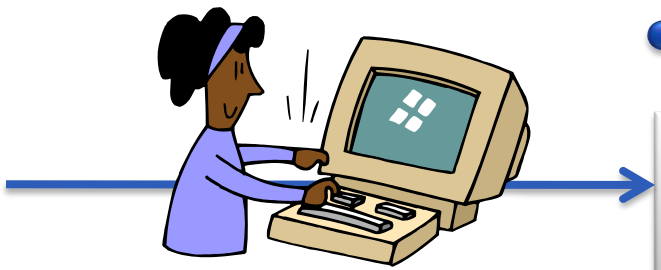
Code is **formal**

Problem in All Software Engineering

Users and customers

Requirements: Usually **informal,** allows many code solutions

Consistent with wishes?



You

Code is **formal**

Problem in All Software Engineering

Users and customers

Feedback

Deployed code

Consistent with wishes?

Requirements: Usually **informal,** allows many code solutions

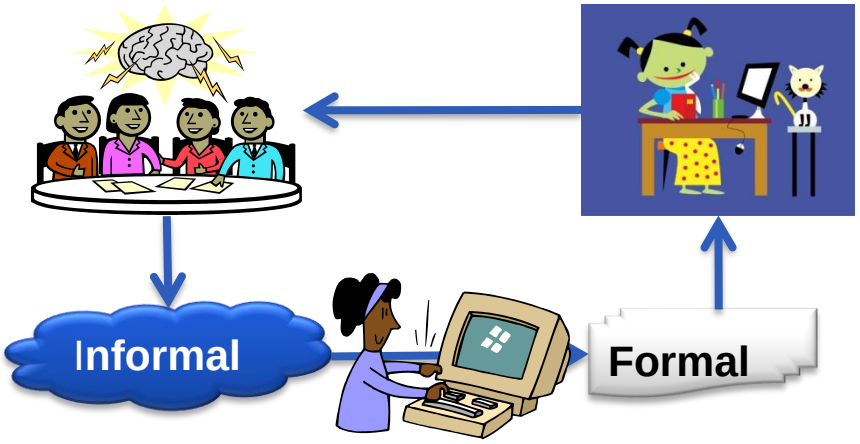
You

Code is **formal**

Problem in All Software Engineering

- Users and customers think in real world terms
  - Money, emotions, quality of live, etc.

- Users and customers
  - Don't think in code (C#, ..)
  - Don't think in UML
  - Don't think in formal logic



- Code has to be in JavaScript, C#, Java, etc.
- You have to bridge that gap

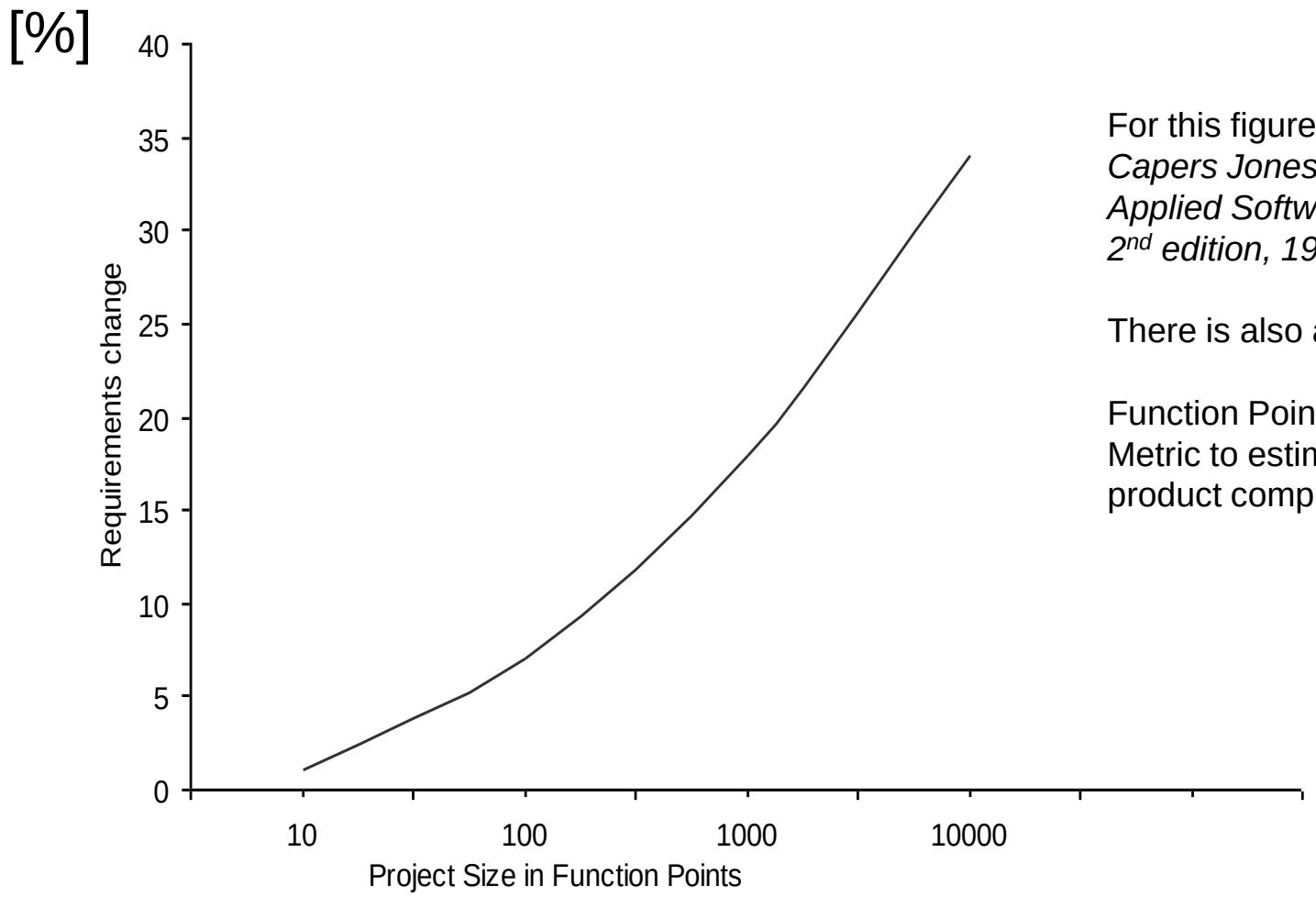
Problems of Classic Process

- Too many requirements: Working in the requirements phase, it is easy to get carried away
  - 45% of requirements were never used [CL p. 23]
  - **Classic process may discover this after delivery (!)**

- Hard to read & understand the requirements
  - People often cannot give good feedback on requirements
  - Stay focused for hundreds of pages of details?
  - Simulate in your mind how these details will work together?
  - **Making customer and users "sign off" on requirements does not solve this problem**

Lot of Change in Large Projects

- ≥ 25% of req. change in large projects [CL p. 24]



For this figure, CL cites:
Capers Jones,
Applied Software Measurement,
2nd edition, 1997

There is also a 3rd edition (2008)

Function Points =
Metric to estimate
product complexity

CL Figure 2.3

Problems of Classic Process

- Requirements change late in the process triggers a lot of rework
  - No matter where change request comes from
  - Adjust all derived (downstream) artifacts to the new requirements: Architecture, design, code, test, manual, ..

What If Not Okay At Last Step?

What is the problem to be solved?

- Describe domain
- Describe requirements
- Review – if okay move to next step

How should we solve the problem?

- High-level: architecture
- Low-level: design
- Review – if okay move to next step

Actually solve the problem

- Write code
- Review, analyze, test, demo **– if okay: done**

Idea: Mitigate With Risk Management

1. What is currently our biggest risk exposure (RE)?
2. Address that risk
3. Repeat

- Major risk: **We are not doing the right thing**
  - According to users'/customers' wishes/needs
  - This is to be expected! (short of brain-reading and users/customers having fully formal brains)
- Address this by implementing **Must-have features first** (MoSCoW) and getting feedback on those
  - After that, users/customers are already somewhat happy as the key features are implemented

Result: Iterative Process

What is the problem to be solved?

- Describe domain
- Describe requirements
- Review

How should we solve the problem?

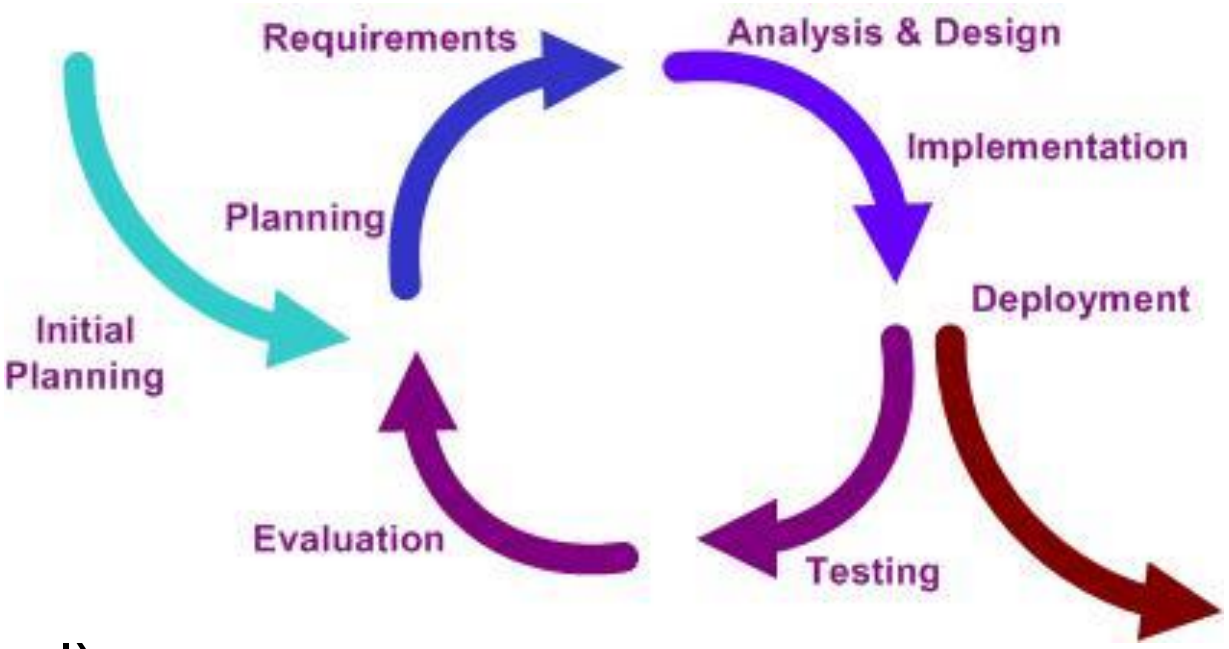
- High-level: architecture
- Low-level: design
- Review

Actually solve the problem

- Write code
- Analyze & Test

Iterative Process

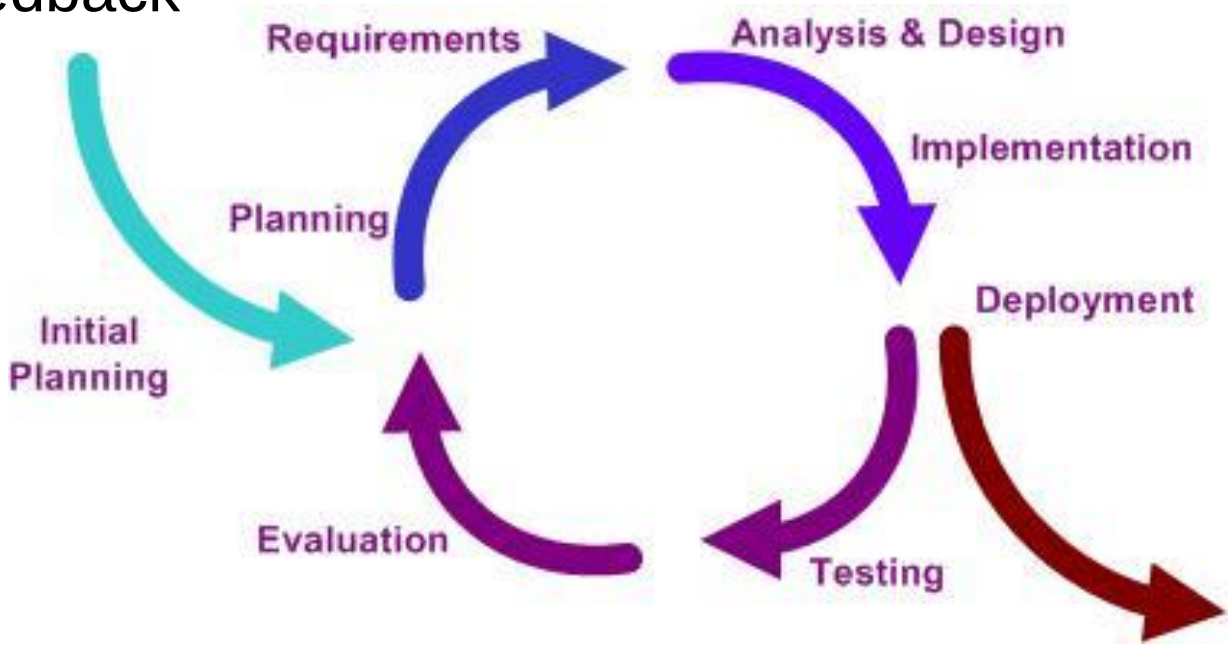
- Iterative or “incremental” or “evolutionary”
- Project is a sequence of short iterations
- Initial planning, then:
- Each iteration is a “mini-project”
  - Planning
  - Requirements
  - Design
  - Code
  - Unit-Test
  - Integrate
  - System-Test
  - Release (internal)



http://en.wikipedia.org/wiki/File:Iterative_development_model_V2.jpg

Each iteration is short

- Quickly get a running system
  - After first iteration
- After that, always have a running system
  - Always ready to demo to get feedback!
  - Always get feedback
  - Always demo



http://en.wikipedia.org/wiki/File:Iterative_development_model_V2.jpg

Each iteration is of fixed length

- “Time-boxed”
- Example: Each iteration is two weeks long, from Monday of week X to Friday of week X+1

- If you cannot finish all requirements in given iteration, push them into future iteration

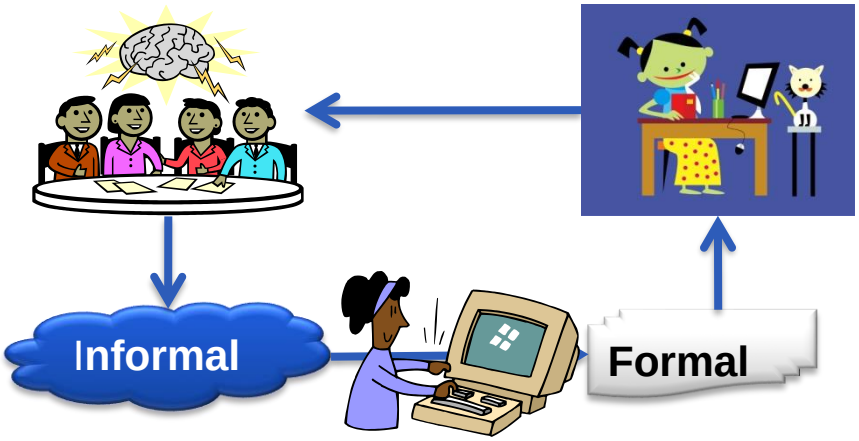
- [+] Allows you to always have a running system
- [+] Forces you to prioritize

Good Risk Management

- Requirements / tasks not scheduled randomly
  - Prioritize tasks according to their risk exposure RE
- The higher the RE, the earlier resolve the risk
- Examples:
  - Need to use new, unproven technology X
    → Plan, design, implement a proof-of-concept of X early
  - System will not work without Y
    → Find out early if Y will work
  - User will hate entire system if Z does not work
    → Plan, design, and implement Z early

Benefits of Iterative Process

- Address and resolve highest risks early
- Some requirement changes are now very easy
  - Have not yet started to design/code/test/debug them!
- Early and frequent feedback from running system



Ideas for Questions to Ask Users

- Explain idea / show prototype
  - What do you like about this concept?
  - How could we change it to make it more valuable to you?

- Let user perform task with your prototype
  - What was difficult about completing this task?
  - How satisfied are you with this feature? (Scale 1..5)
  - How likely are you to recommend us to a friend? (1..10)

- Ideally observe user's body language
  - In-person / video conference are better than online form

[https://future.com/user-research-as-you-grow/]

How to

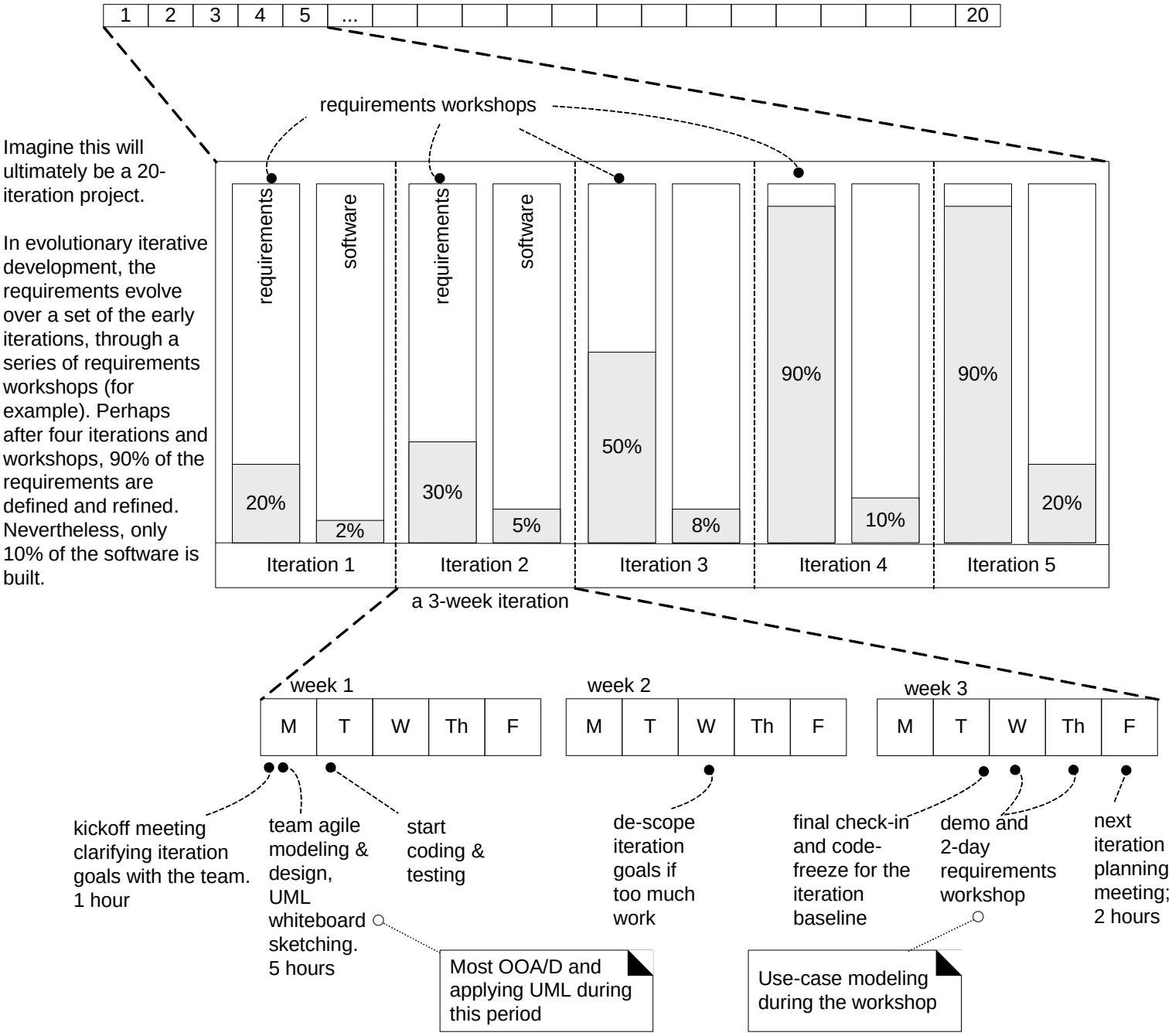
- Before iteration 1
  - Gain high-level overview of requirements
  - Pick small subset X = highest technical and business risk
  - Analyze requirements of X
- Iteration 1: Design, code, test part of X
- More planning in early iterations
  - Overview of all requirements, analyze requirements
  - Schedule requirements into early or late iterations
- More coding and testing in later iterations
  - Less need for planning
- Next slide gives an example for 20 iterations

IN-CLASS EXERCISE: SCHEDULE HIGH-RE TASKS

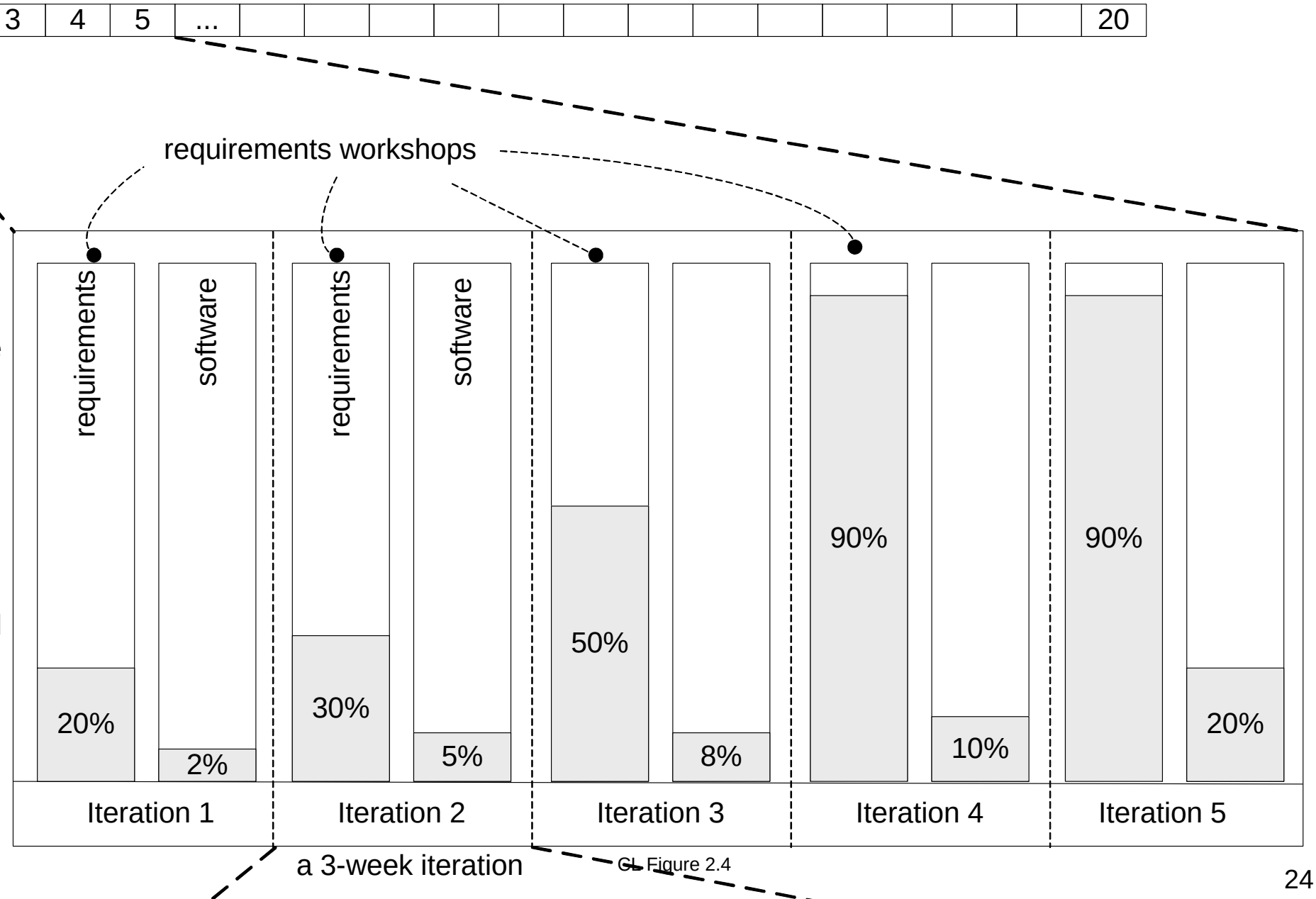
Schedule High Risk Exposure Tasks

- Get together with your team (breakout room)
- Update the risk exposure of your project's top risks
  - Risk exposure :=
    (Probability of risk turning real) *
    (Effect of risk turning real)
- Create tasks to address the top risks
- Add these tasks to your iteration plan

- Post your updated risks & iteration plan in the Teams chat

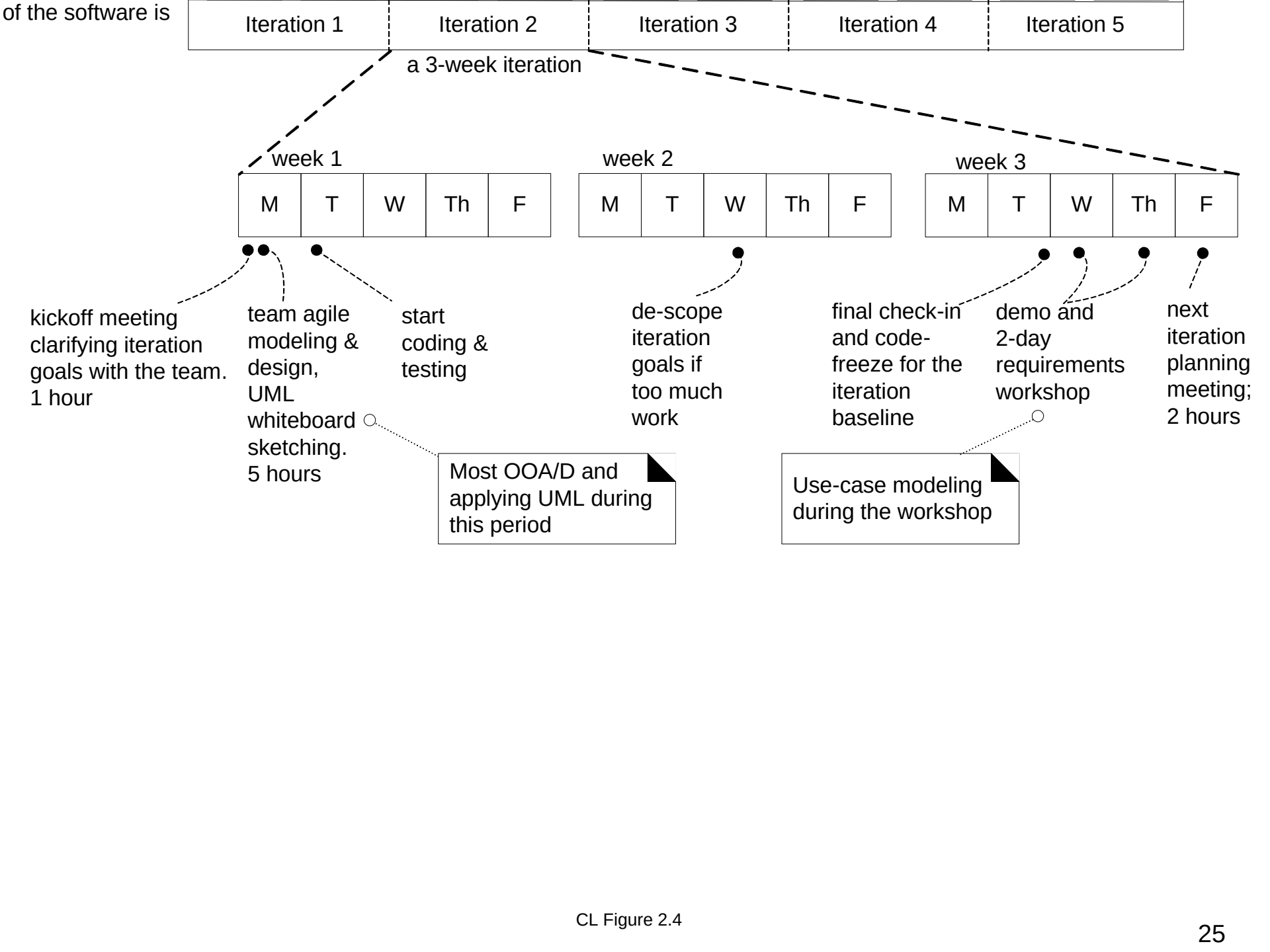

1
2
3
4
5
...
20
requirements workshops
Imagine this will ultimately be a 20-iteration project.
In evolutionary iterative development, the requirements evolve over a set of the early iterations, through a series of requirements workshops (for example). Perhaps after four iterations and workshops, 90% of the requirements are defined and refined. Nevertheless, only 10% of the software is built.
requirements
software
requirements
software
20%
2%
30%
5%
50%
8%
90%
10%
90%
20%
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
a 3-week iteration
week 1
week 2
week 3
M
T
W
Th
F
kickoff meeting clarifying iteration goals with the team. 1 hour
team agile modeling & design, UML whiteboard sketching. 5 hours
start coding & testing
de-scope iteration goals if too much work
final check-in and code-freeze for the iteration baseline
demo and 2-day requirements workshop
next iteration planning meeting; 2 hours
Most OOA/D and applying UML during this period
Use-case modeling during the workshop

CL Figure 2.4


3
4
5
...
20
requirements workshops
requirements
software
requirements
software
20%
2%
30%
5%
50%
8%
90%
10%
90%
20%
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
a 3-week iteration

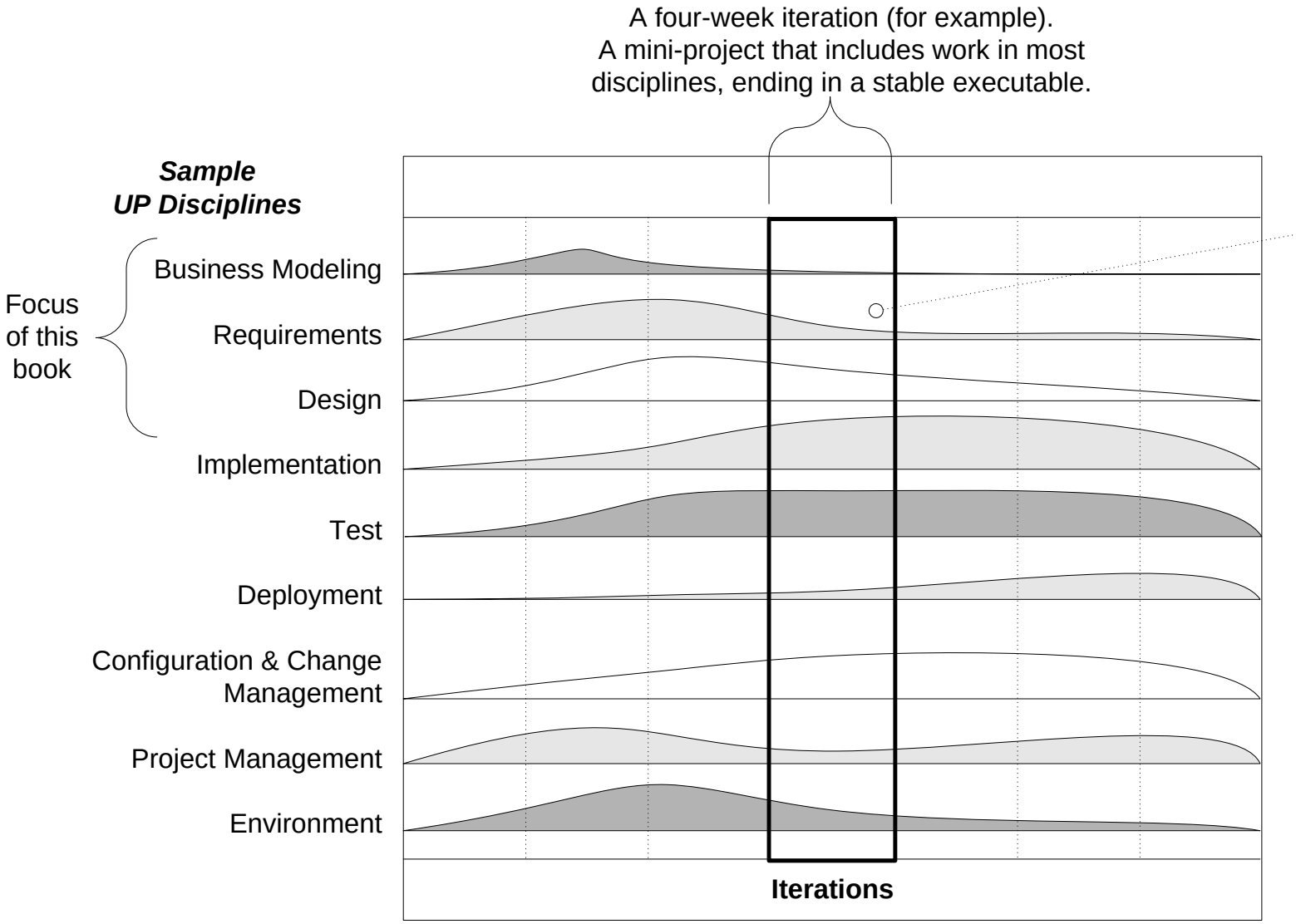
GL Figure 2.4

of the software is



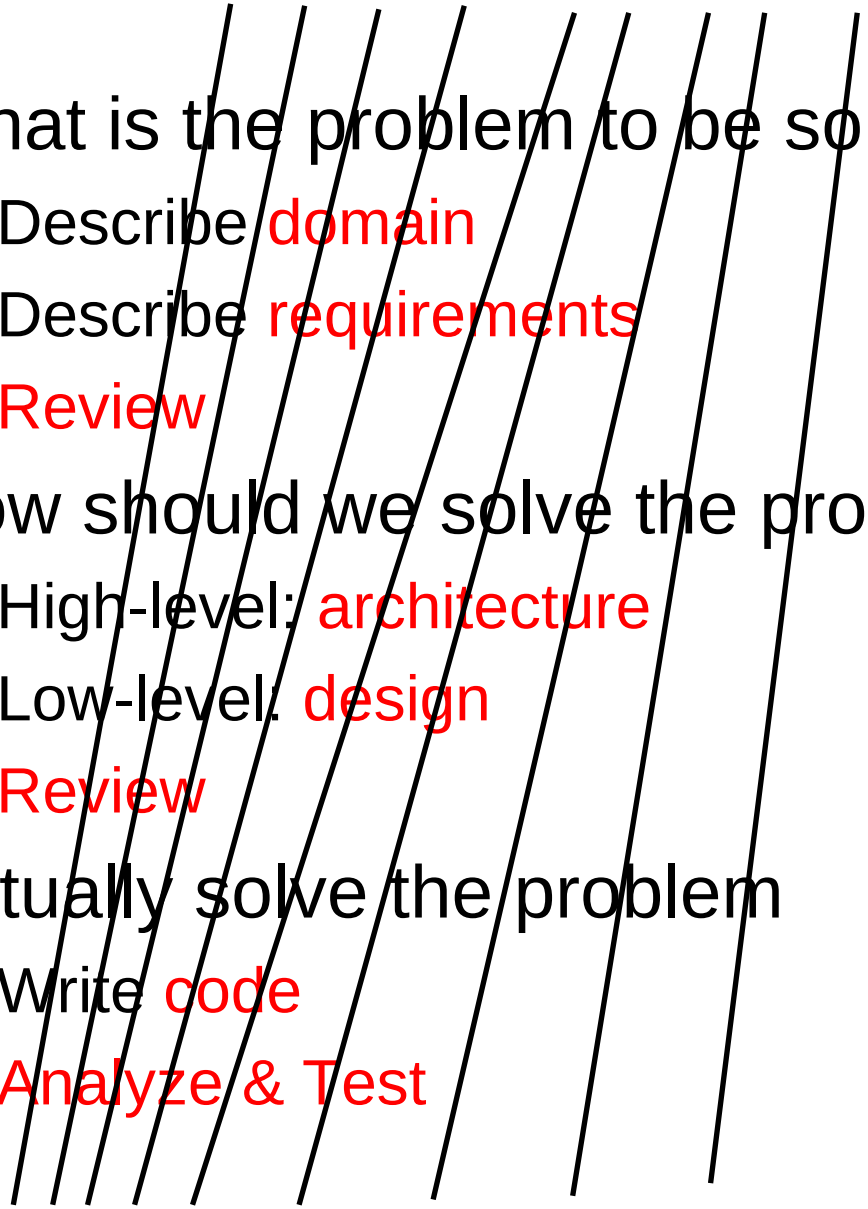
CL Figure 2.4

Requirements + Software + ...



CL Figure 2.7

Line Picture: Iteration 2 ☺



What is the problem to be solved?

- Describe domain
- Describe requirements
- Review

How should we solve the problem?

- High-level: architecture
- Low-level: design
- Review

Actually solve the problem

- Write code
- Analyze & Test

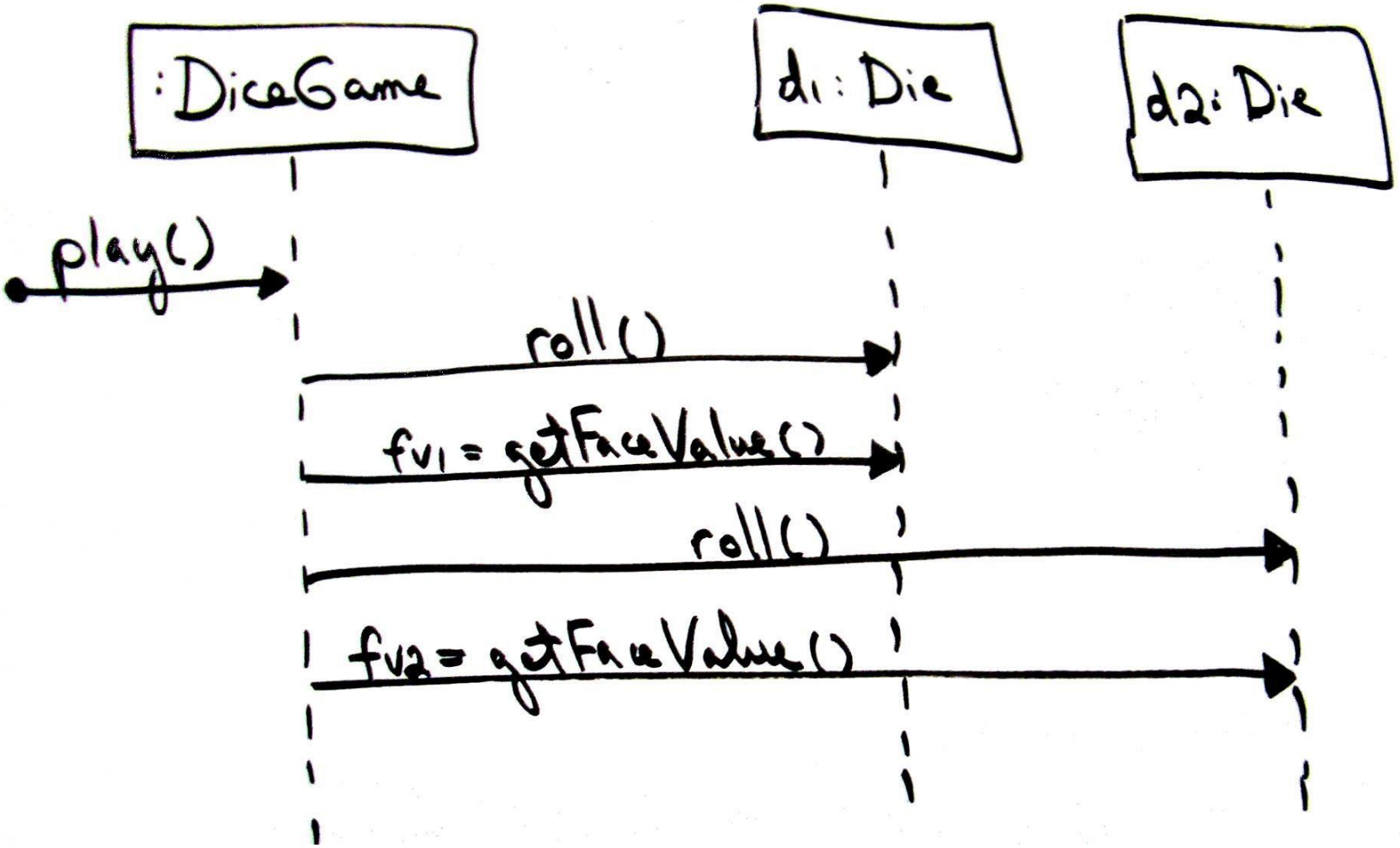
Agile Modeling

- Easier to play with UML model than with full code
  - Use modeling to save yourself time

- ***Model to understand****, not to document everything*
  - You are paid for quality code, not pretty UML diagrams

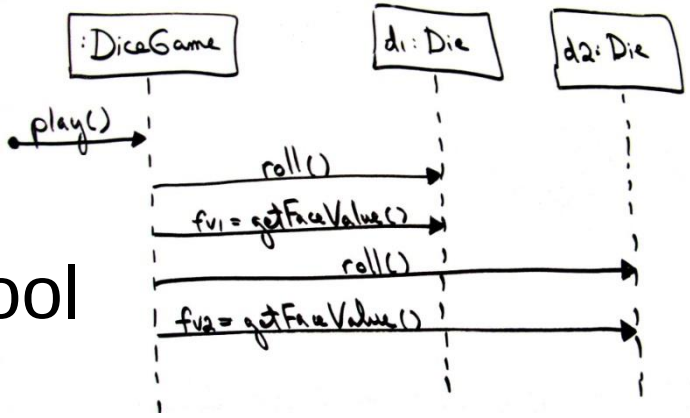
- Capture and preserve that understanding!
  - Easy to forget after a few weeks/months
  - Dedicated meeting room: Keep UML models on walls
  - Share picture of paper-based sketch with smartphone
  - Collaborative online sketch (Teams Whiteboard, ..)

Example: UML Sequence Diagram



CL Figure 1.4

Example



- Quick and dirty
- Not with formal UML modeling tool

- It may work just fine to
  - Use whiteboard & camera instead of a modeling tool
  - See mock screenshot prototyping with paper/pencil

- Wait, but what about code generation?
  - Will not work from a jpeg ;-)
  - But how much will the fancy UML modeling tools do?
  - I have not seen a good tool yet.
  - But if you have one, by all means use it!

UML to code: James Rumbaugh

- One of the main forces behind UML (Rational Software)
- Q: What do you think of using UML to generate implementation code?
- James Rumbaugh:
  - “I think it is a terrible idea. I know that I disagree with man other UML experts, but there is no magic about UML.
  - **If you can generate code from a model, then it is a programming language.**
  - **And UML is not a well-designed programming language.** [..]
  - I think a lot of the recent work on UML has been misguided. It never was meant to be a programming language. Use it to get the strategy right and write the final program in a suitable programming language.”

[Masterminds, page 339, formatted]

UML vs. code: James Rumbaugh

- Q: Is UML just a tool to coordinate work in a large team of developers?
- James Rumbaugh:
  - "It is first and foremost a tool for guiding and organizing the thinking of individual developers.
  - **You need to work at different levels of abstraction; code is a particular level, but not the most useful level for understanding how systems work.**
  - You need to work at a higher level, which means getting away from all the details of the code to the things that are important at the higher level. That's why it's a mistake to make UML executable; that would destroy the whole point of abstraction."

    [Masterminds, page 340, formatted]

Agile Modeling Tips

- Don’t model alone
  - Others will understand more if they helped modeling
- Maintain a positive, improvement-oriented attitude
  - Discover, share, understand, don’t fight ;-)
- Model different aspects of same thing in parallel

- Use the simplest notation possible
  - Use only basic UML elements
- Non-basic UML: Justify & give good references
  - Otherwise others may misunderstand you!
  - You may later even misunderstand yourself ;-)