COMP20050 - Software Engineering Project II

Team Work, Agile Software Development Process

Ravi Reddy Manumachu
ravi.manumachu@ucd.ie

UCD School of Computer Science.

Scoil na Ríomheolaíochta UCD.

Outline

- The Discipline of Teams.
- Software Engineering Concepts Brief.
- Plan-based Software Development Approach.
- Agile Software Development Approach.

Resources

- **The Discipline of Teams**
  https://hbr.org/1993/03/the-discipline-of-teams-2

- **Ground Rules for Effective Groups**
  By Roger Schwarz
  https://brightspace.ucd.ie/d2l/le/content/249619/viewContent/2904386/View

- **What is Agile?**
  https://www.agilealliance.org/agile101/

- **Essential Scrum: A Practical Guide to the Most Popular Agile Process**
  By Kenneth S. Rubin
  Publisher: Addison-Wesley Professional; 1st edition

Team Work

What is a Team?

A **team** is a small number of people with **complementary skills** who are committed to a **common purpose**, set of **performance goals**, and approach for which they hold themselves **mutually accountable**.

(**Jon R. Katzenbach and Douglas K. Smith**, Harvard Business Review, 1993)

Teams and Working Groups

- **Teams** differ fundamentally from **working groups** because they require both individual and mutual accountability.
- **Teams** produce discrete work-products through the joint contributions of their members.
- For example, you as a **team** should **jointly**
  - **Design** the system components (abstractions) and their interaction in the Black Box+ Board game software system.
  - **Design** the data structures/classes for **BlackBoxBoard**, **Atom**, **CircleOfInfluence**, **RayPath**, and **RayMarker**.
  - **Formulate** the **project plan**.

Essential Characteristics of Teams

- A meaningful **common purpose** that the team has helped shape.
- Specific **performance goals** that flow from common purpose.
- A mix of **complementary skills**.
- A **strong commitment** to how the work gets done.
- **Mutual accountability**.
- **Psychological safety** (shared belief that the team is safe for interpersonal risk taking).
- Open, explicit communications, willingness to "**discuss the undiscussable**".

Building Team Performance

- Establish **urgency**, demanding performance standards, and **direction**.
- *Select members for skill and skill potential, not personality. (Not applicable here)*
- Pay particular attention to **first meetings and actions**.
- Set some **clear rules of behavior**.
- Set and seize upon a few **immediate performance-oriented tasks and goals**.
- Challenge the group regularly with **fresh facts and information**.
- Spend lots of **time together**.
- Exploit the power of positive **feedback**, **recognition**, and **reward**.

Ground Rules

Ground Rules for Effective Groups

- A group can be **ineffective** even when it has clear goals and talented and motivated members.
- One reason is **lack of ground rules.**
- **Many effective groups have explicit ground rules that guide their behavior.**

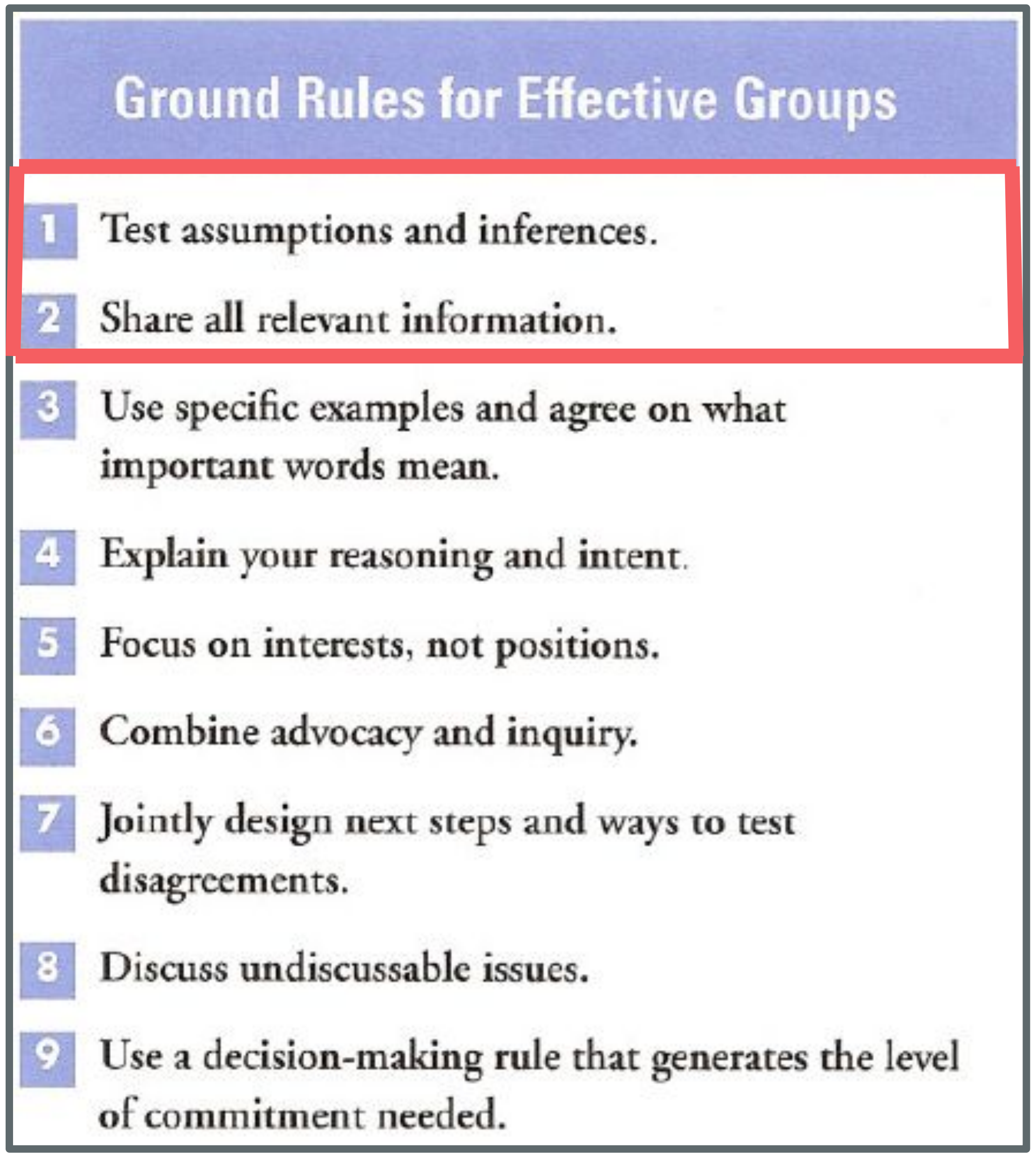
Four Core Values

- The ground rules are based on four core values.
- Groups need **valid information** to effectively solve problems and make decisions.
- With **free and informed choice**, group members make their decisions based on valid information.
- With **internal commitment**, each member feels personally responsible for the decision and is willing to support the decision, given his or her role.
- With **compassion**, you temporarily suspend judgment to understand others who have differing views.

Four Assumptions

- There are four **assumptions** behind the ground rules.
    - I have some relevant information, and other people also have relevant information.
    - Each of us may see things the others do not.
    - Differences are opportunities for learning.
    - People are trying to act with integrity, given their situation.

Ground Rules 1 & 2

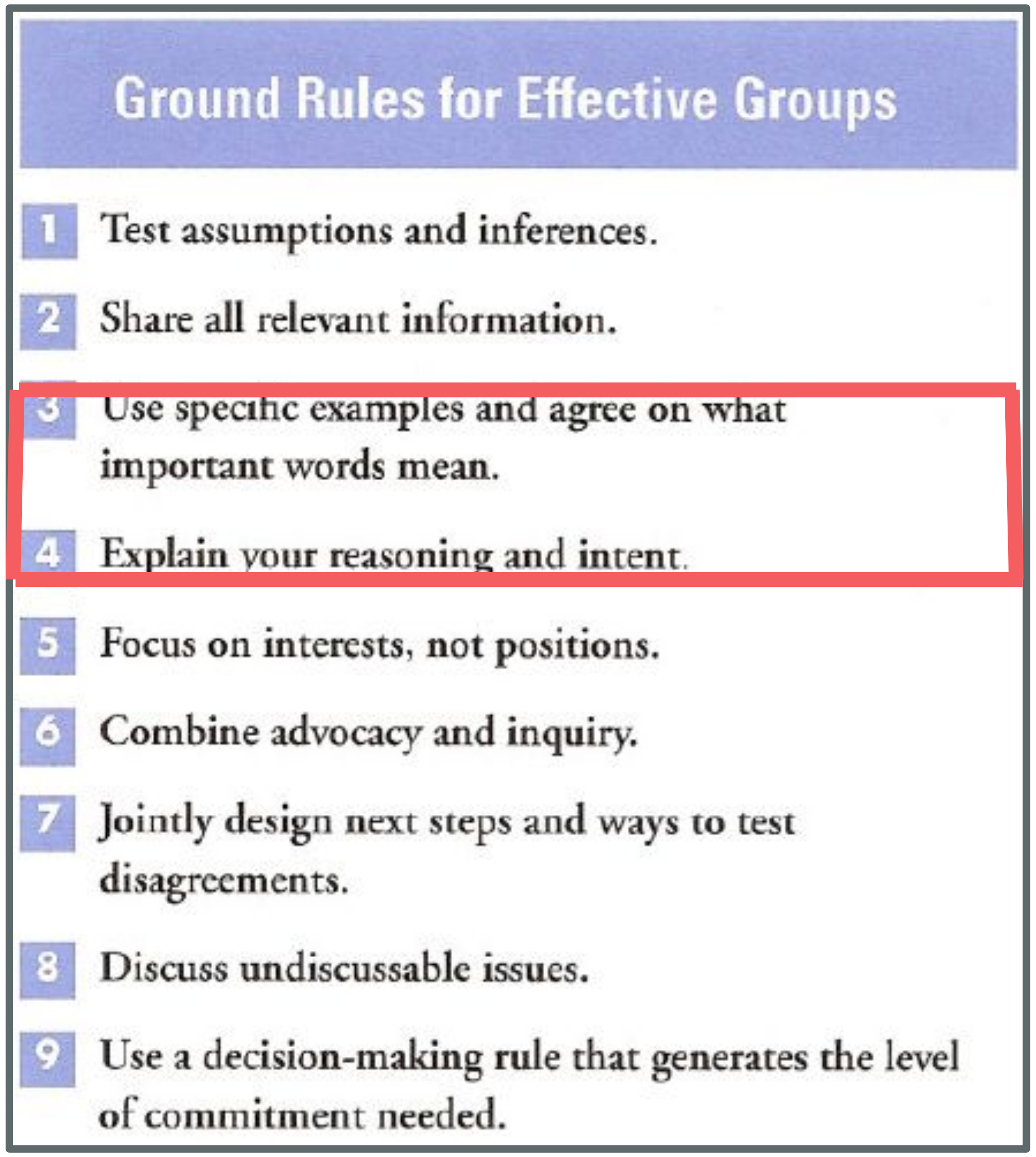

Ground Rules for Effective Groups

1. Test assumptions and inferences.
2. Share all relevant information.
3. Use specific examples and agree on what important words mean.
4. Explain your reasoning and intent.
5. Focus on interests, not positions.
6. Combine advocacy and inquiry.
7. Jointly design next steps and ways to test disagreements.
8. Discuss undiscussable issues.
9. Use a decision-making rule that generates the level of commitment needed.

- **GR 1:** When you don't **test your inferences and assumptions** with the people about whom you are making them, you may be basing your actions on a set of conclusions that are completely flawed.

- **GR 2:** **Share and not withhold all relevant information** you have that might affect how the group solves a problem or makes a decision.

Ground Rules 3 & 4



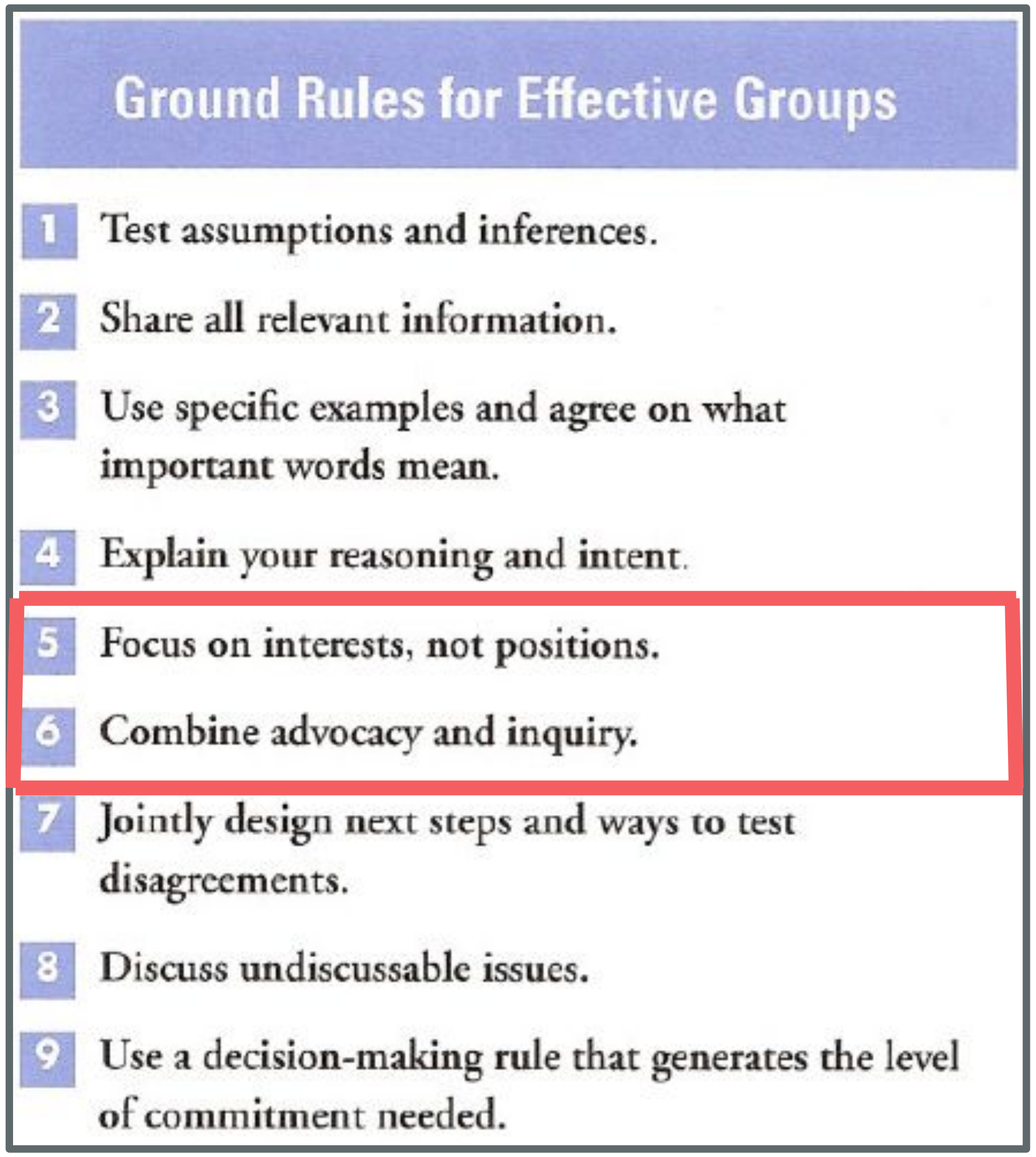
Ground Rules for Effective Groups

1. Test assumptions and inferences.
2. Share all relevant information.
3. Use specific examples and agree on what important words mean.
4. Explain your reasoning and intent.
5. Focus on interests, not positions.
6. Combine advocacy and inquiry.
7. Jointly design next steps and ways to test disagreements.
8. Discuss undiscussable issues.
9. Use a decision-making rule that generates the level of commitment needed.

- **GR 3: Using specific examples and agreeing on what important words mean** is one way of sharing relevant information, generating valid data and creating a common understanding.

- **GR 4: Explaining your reasoning and intent** includes making your private thinking public. It enables others to see how you reached your conclusions.

Ground Rules 5 & 6



Ground Rules for Effective Groups

1 Test assumptions and inferences.

2 Share all relevant information.

3 Use specific examples and agree on what important words mean.

4 Explain your reasoning and intent.

5 Focus on interests, not positions.

6 Combine advocacy and inquiry.

7 Jointly design next steps and ways to test disagreements.

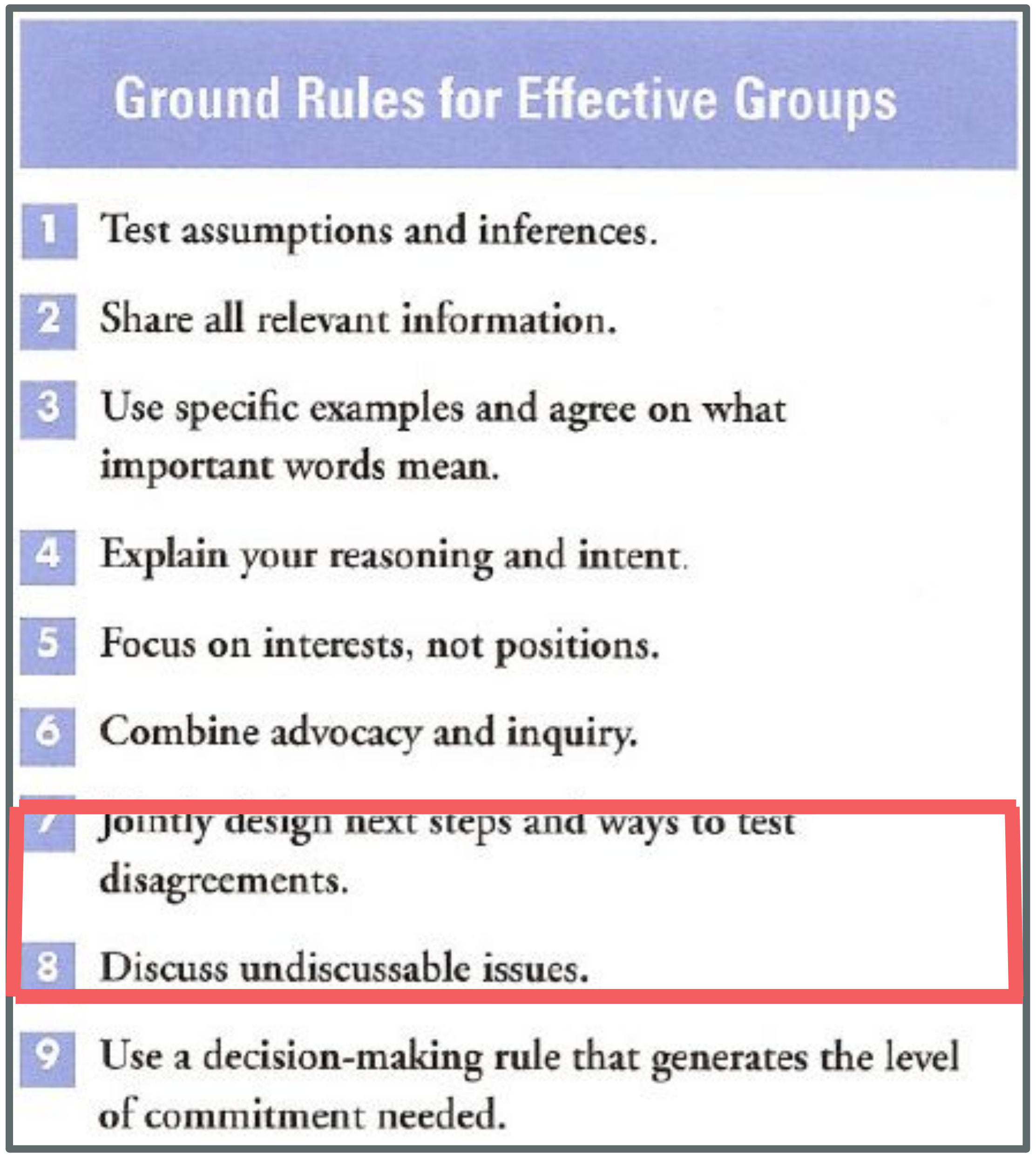
8 Discuss undiscussable issues.

9 Use a decision-making rule that generates the level of commitment needed.

- **GR 5:** To help the group **focus on interests rather than positions**, you might begin by asking each member to list the criteria that must be met in order for that member to accept a solution.

- **GR 6: Combining advocacy and genuine inquiry** shifts a meeting from a series of unconnected monologues to a focused conversation and creates conditions for learning.

Ground Rules 7 & 8

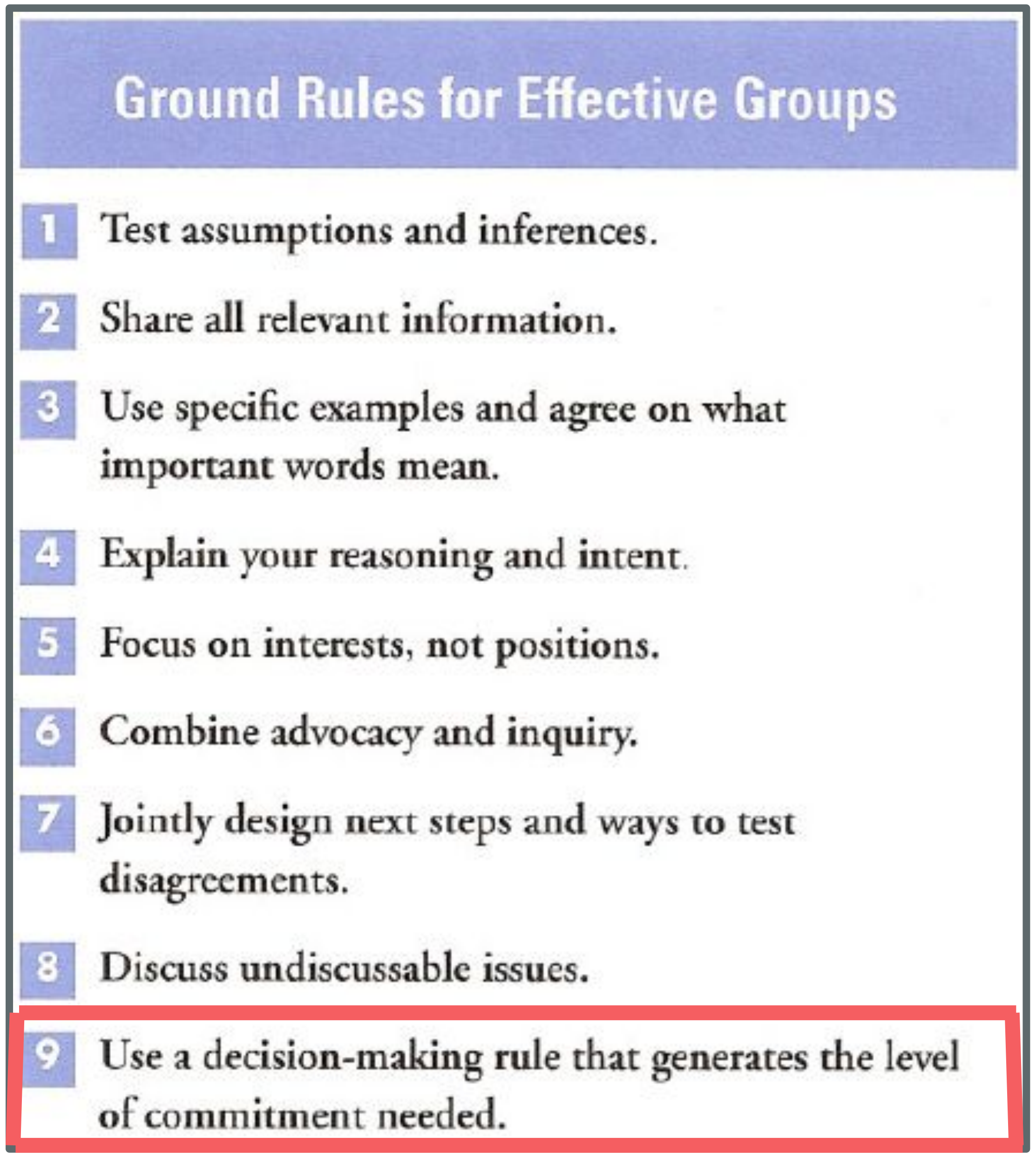

Ground Rules for Effective Groups

1. Test assumptions and inferences.
2. Share all relevant information.
3. Use specific examples and agree on what important words mean.
4. Explain your reasoning and intent.
5. Focus on interests, not positions.
6. Combine advocacy and inquiry.
7. Jointly design next steps and ways to test disagreements.
8. Discuss undiscussable issues.
9. Use a decision-making rule that generates the level of commitment needed.

- **GR 7: Jointly designing next steps and ways to test disagreements** means deciding with others what topics to discuss, when and how to discuss them and when to switch topics, rather than making these decisions privately and unilaterally.

- **GR 8:** Follow all the previous ground rules to use this ground rule since it is emotionally difficult to use.

Ground Rule 9


Ground Rules for Effective Groups

1. Test assumptions and inferences.
2. Share all relevant information.
3. Use specific examples and agree on what important words mean.
4. Explain your reasoning and intent.
5. Focus on interests, not positions.
6. Combine advocacy and inquiry.
7. Jointly design next steps and ways to test disagreements.
8. Discuss undiscussable issues.
9. Use a decision-making rule that generates the level of commitment needed.

- **GR 9:** It makes specific the core value of internal commitment.

Software Engineering Concepts

Requirements Engineering

- The **requirements** for a system are the descriptions of what the system should do, the services that it provides and the constraints on its operation.

- The process of finding out, analyzing, documenting and checking these services and constraints is called **requirements engineering** (RE).

Two Levels of Description

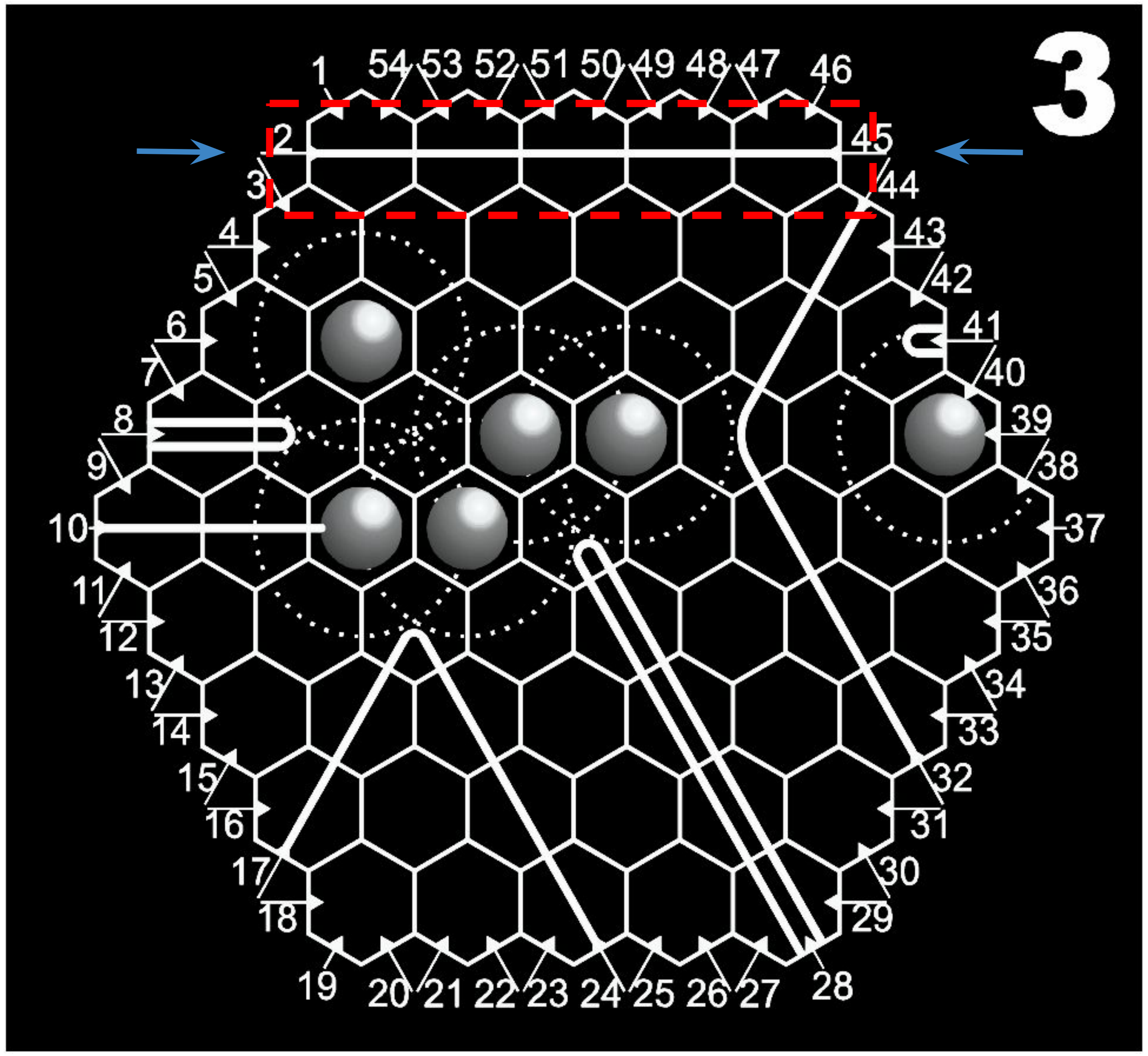
- **User requirements** are statements (natural language plus diagrams), of what services the system is expected to provide to system users and the constraints under which it must operate.

- **System requirements** are more detailed descriptions of the software system's functions, services, and operational constraints. (all system requirements go into **functional specification**).

Two Types of System Requirements

- **Functional requirements** are statements of services the system should provide.
- **Non-functional requirements:**
  - Are constraints on the services or functions offered by the system.
    - Performance, security, availability, scalability, Installability (and many more).
  - Are often more critical than individual functional requirements.
  - Failing to meet a non-functional requirement can render the whole system unusable.
- **Non-functional requirements** for **Black Box+** software system.
  - Portability, Installability, Maintainability.

Black Box+ User Requirements



…

- **URx:** The Black Box Software System (BBSS) shall record a black marker for a hit.

…

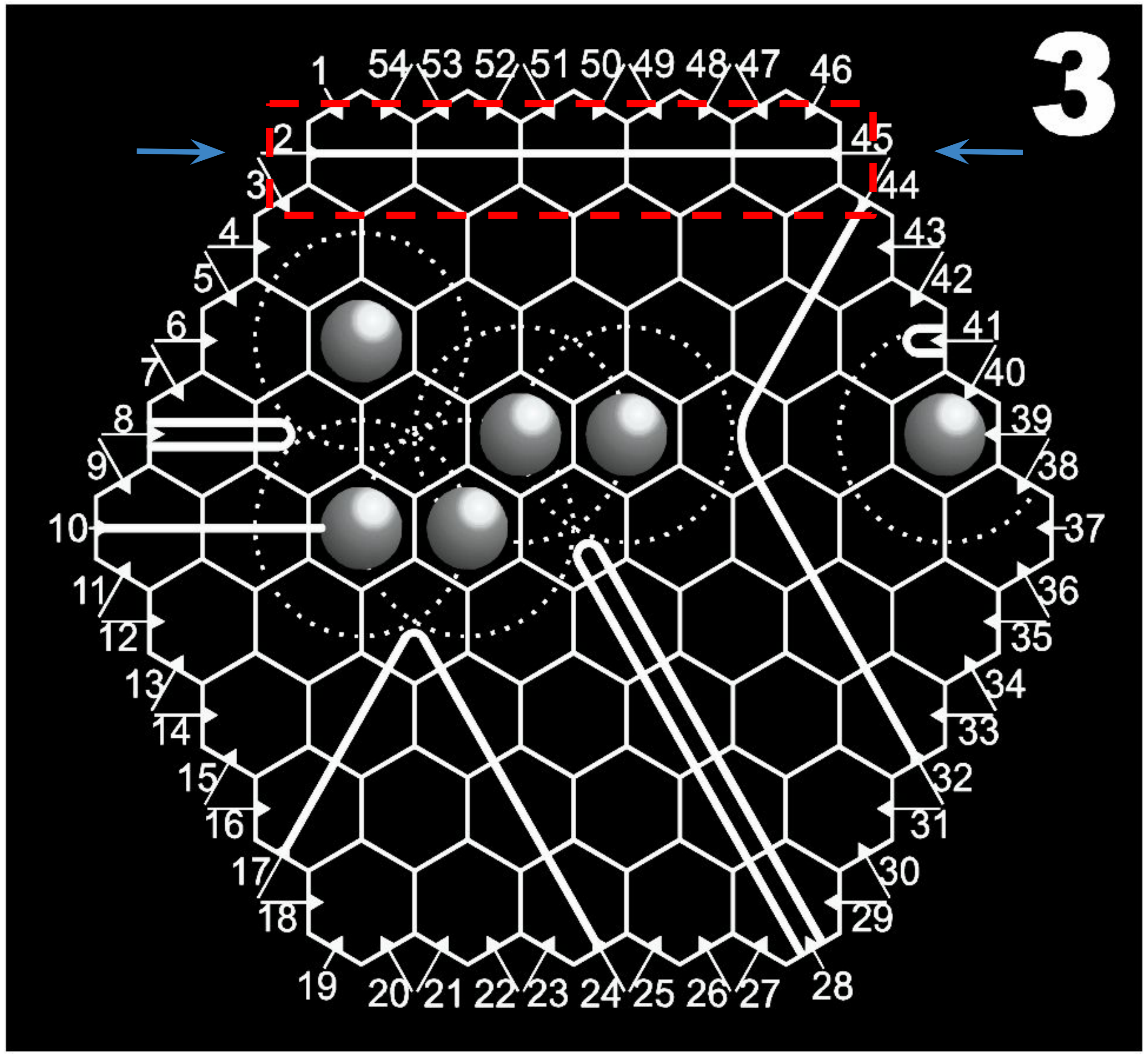
- **URy:** The Black Box Software System (BBSS) shall record a white marker for a reflection.

…

- **URz:** The Black Box Software System (BBSS), for a ray input at edge number 2 by the experimenter in the accompanying figure, shall record a pair of ray markers of the same colour (not black or white) at edge number 45.

…

Black Box+ System Requirements for URz



- **SRz.1**: The BBSS shall prompt experimenter to input the edge number and the experimenter types 2.
- **SRz.2**: The BBSS shall prompt the setter to announce the result and the setter types 45.
- **SRz.3**: The BBSS shall place a pair of ray markers (red blocks) at the edges 2 and 45.

NOTE: Since the setter in your case is the AI algorithm, SRz.2 and SRz.3 will be combined as follows:

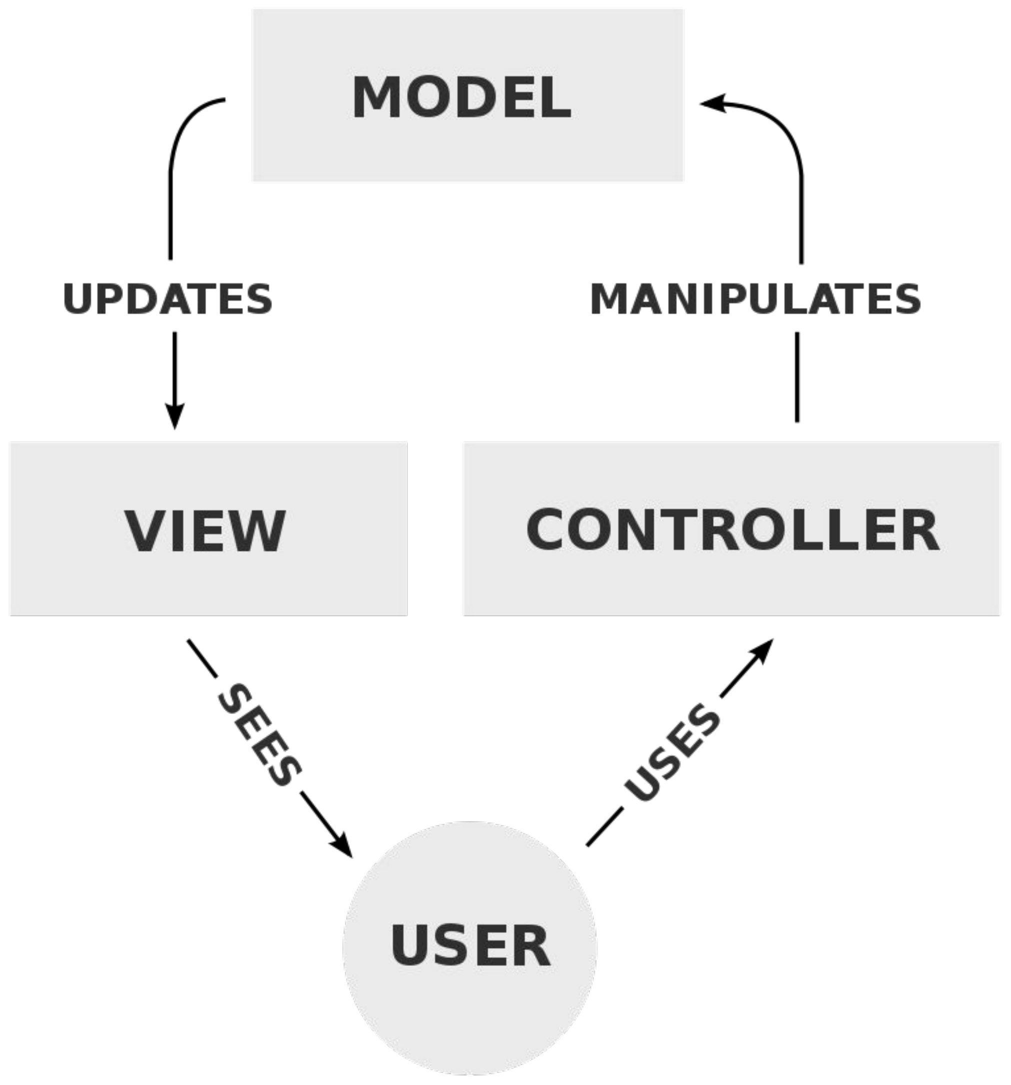
The BBSS shall place a pair of ray markers (red blocks) at the edges 2 and 45.

Software Architectural Design

- **Architectural design** is a creative process where you design a system organization that will satisfy the **functional** and **non-functional** requirements of a system.
- **System architects** must consider some fundamental questions about the system:
  - Is there a **generic application architecture** that can act as a template for the system that is being designed?
  - What **architectural patterns or styles** might be used?
  - What will be the fundamental approach used to structure the system?
  - How will the **structural components** in the system be decomposed into sub-components?

Architectural Pattern

- An **architectural pattern** is an abstract description of good practice, which has been tried and tested in different systems and environments.
- Consider the **Model-View-Controller (MVC)** pattern.



Model-View-Controller (MVC)

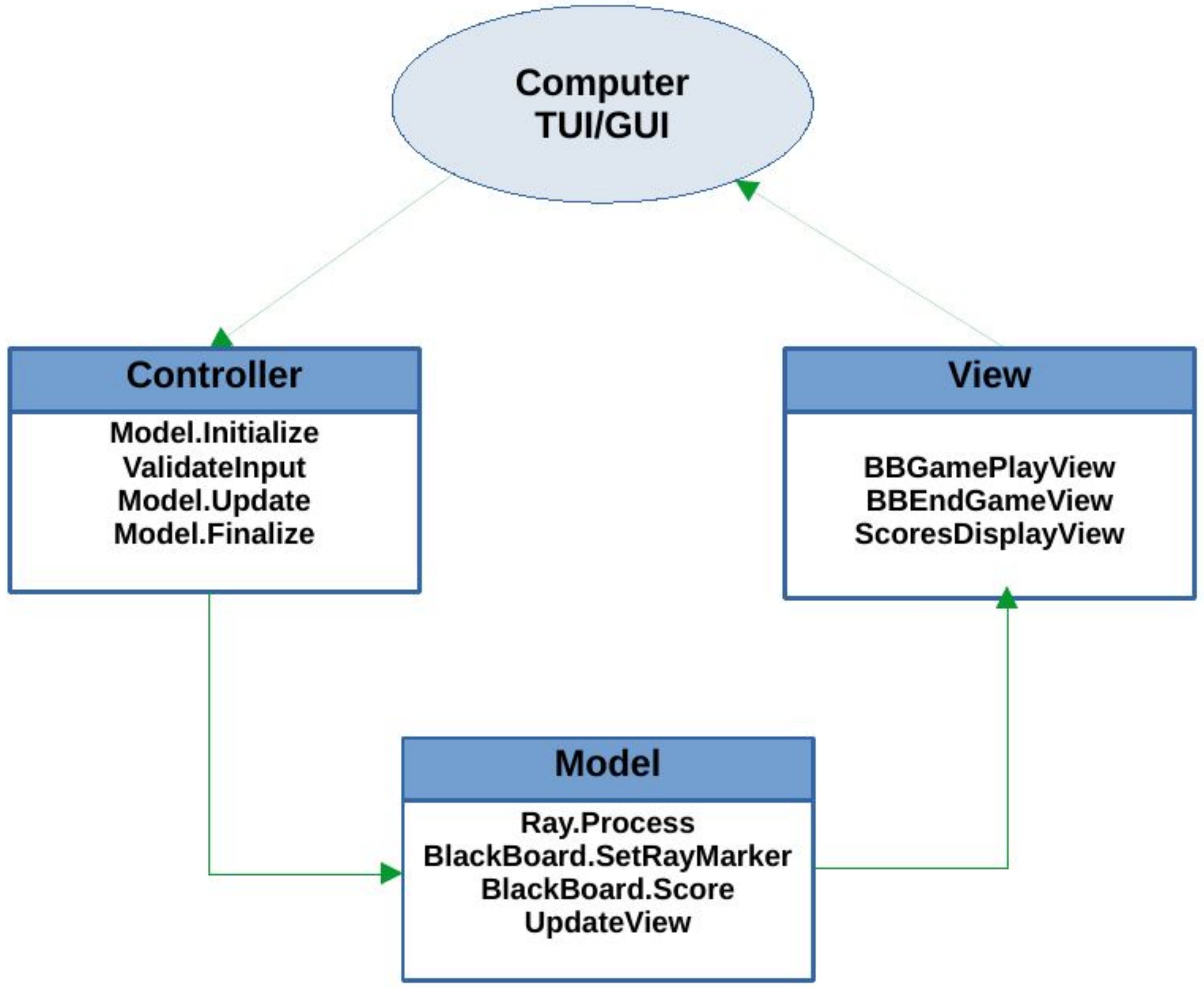
- The **Model** represents the state of the application and any business logic or operations that should be performed by it. For example:
  - **Interfaces/Classes** for BlackBoxBoard, RayPath, RayMarker, Atom, CircleOfInfluence.
  - **Algorithms** (in setter) to trace complex paths of rays.

- **Views** are responsible for presenting content through the user interface. For example:
  - **BlackBoardStartView**, **BlackBoardGamePlayView**, **FinalScoreView**.

- **Controllers** are the components that handle user interaction, work with the model, and ultimately select a view to render (a model can also do this). For example:
  - **ValidateExperimenterInput**, **Model.Update**, **Model.Finalize**.

MVC for Black Box+ Design

- **Model-View-Controller pattern (MVC)** for Black Box+ design.



Software Design and Implementation

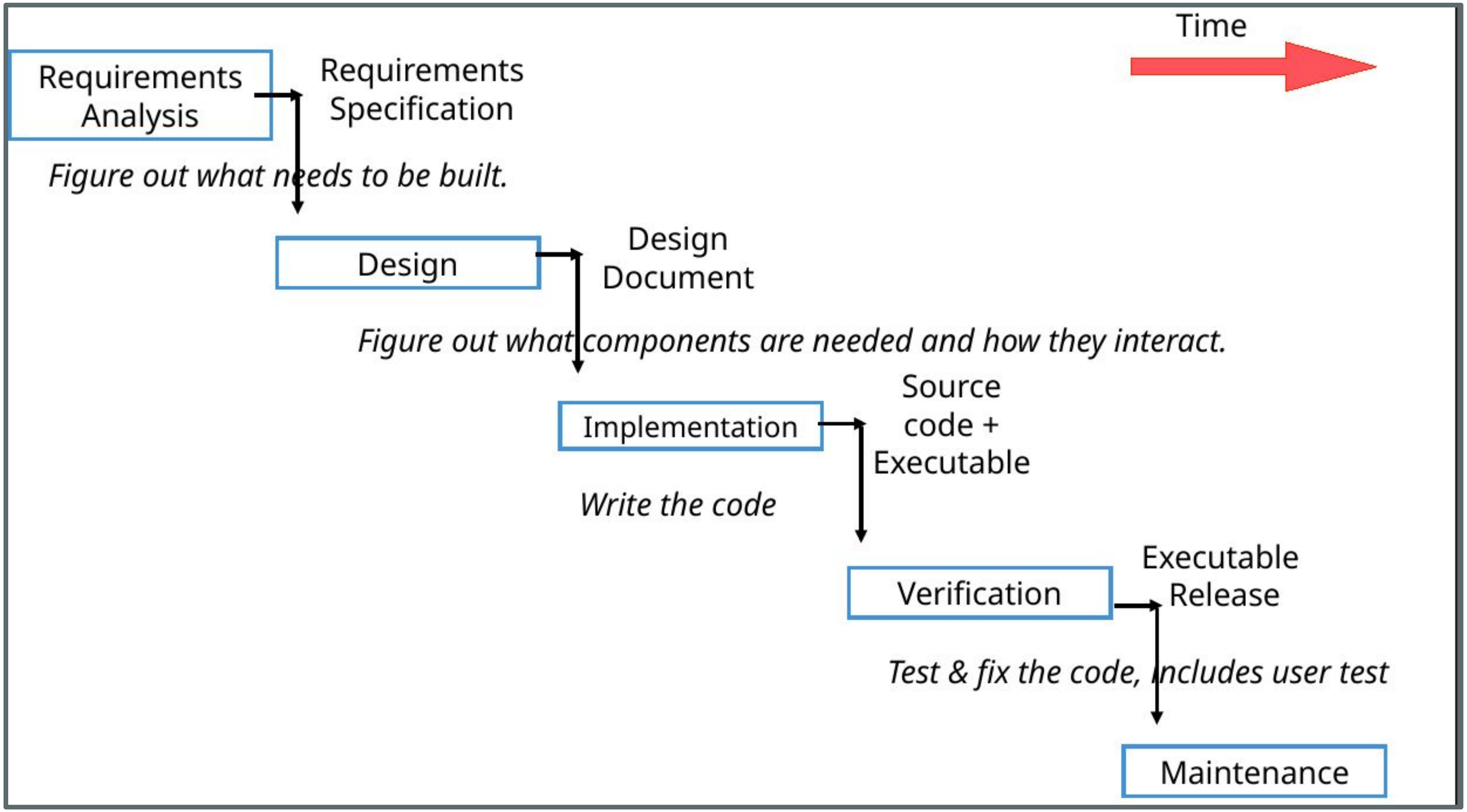
- **Software design and implementation** is the stage in the software engineering process at which an executable software system is developed.

- **Structural models,** which describe the static structure of the system using object classes and their relationships.
  - **BlackBoxBoard**, **Atom**, **CircleOfInfluence**, **RayPath**, **RayMarker**.

- **Dynamic models**, which describe the dynamic structure of the system and show the interactions between the system objects.
  - For example: a **sequence diagram** that shows the interaction between all the above objects when experimenter specifies an edge for the ray entry resulting in a hit.

Traditional Plan-driven Process

Traditional Plan-driven Development


Time
Requirements Analysis
Requirements Specification
Figure out what needs to be built.
Design
Design Document
Figure out what components are needed and how they interact.
Implementation
Source code + Executable
Write the code
Verification
Executable Release
Test & fix the code, includes user test
Maintenance

Traditional waterfall process

Features of Traditional Approaches

- The **waterfall process** falls under the broader class of **plan-driven processes** (also known as **traditional** or **sequential** development processes).

- **Plan-driven processes** are often called **sequential processes** because practitioners perform the following major phases in **software development lifecycle** sequentially:
  - A complete requirements analysis.
  - A complete design.
  - Coding.
  - Testing.
  - Deployment and Maintenance.

Plan-driven Approaches: Drawbacks (1/2)

- Plan-driven development works well for problems that are **well defined**, **predictable**, and **unlikely to undergo any significant change**.

- Plan-driven approaches are based on a set of beliefs that do not match the **uncertainty** inherent in most product development efforts.

- In real-world software development projects, things never go according to plan.

Plan-driven Approaches: Drawbacks (2/2)

- Plan-driven approach is **not flexible** because each phase needs to be fully completed before moving on to the next phase.
- In plan-driven approach, clients and stakeholders are not involved during the process. **Revised stakeholder priorities** can shift development timelines and efforts.
- **Requirements** can become **outdated** before development even begins.
- **New technologies** mean changes to even the best designs.

Agile Process/Methodology

What is Agile?

- **Agile processes** emerged in the late 1990s that shunned the rigid structure of the **Waterfall method** of software development.
- **Agile** is the ability to create and respond to change.
- It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment.

Agile Software Development

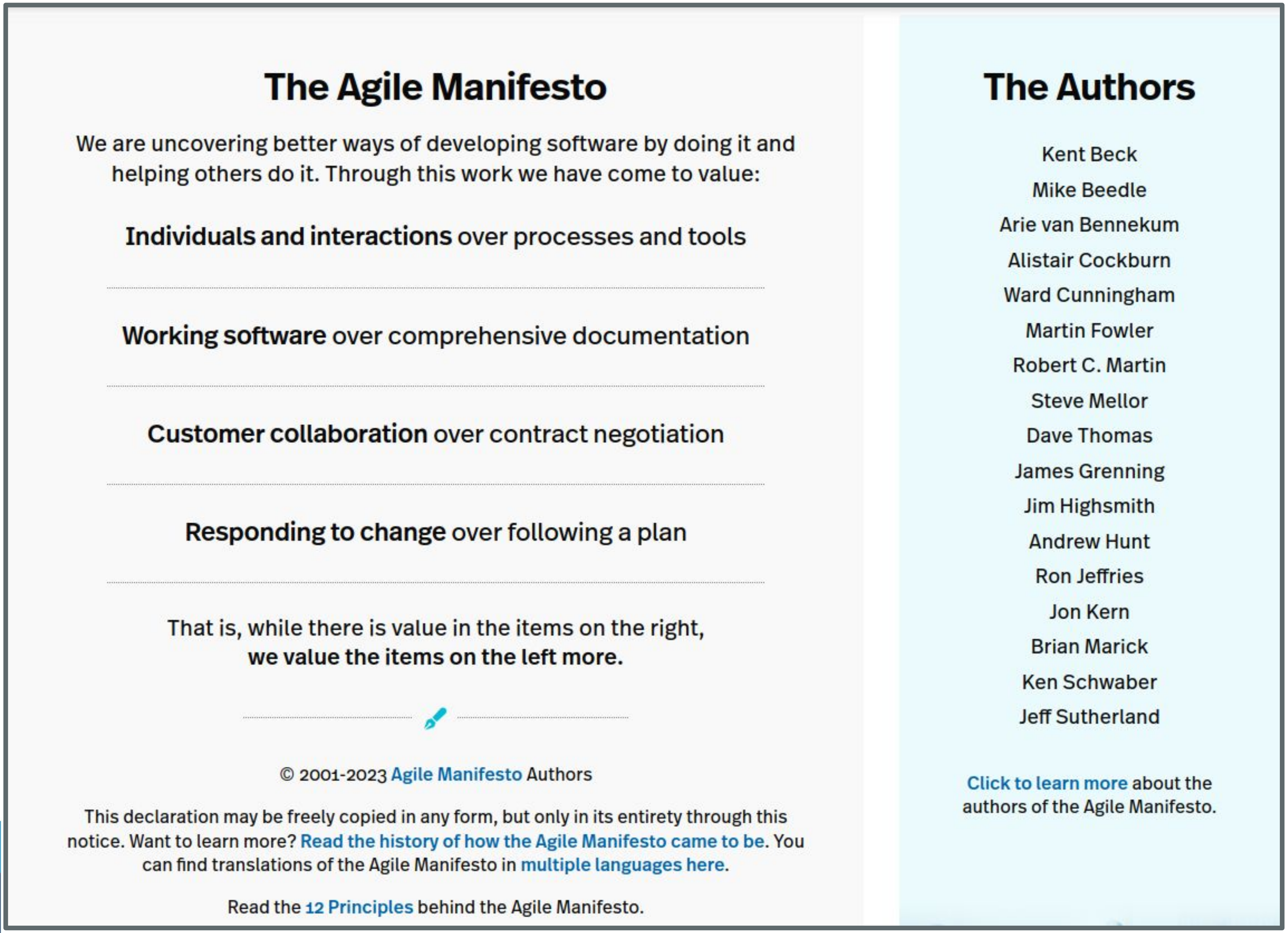
- **Agile software development** is an umbrella term for a set of **frameworks** and **practices** based on the values and principles expressed in the **Manifesto for Agile Software Development.**

- Also known as **rapid/iterative software development process** designed to produce useful software quickly in a series of increments.

- **Frameworks:**
    - **Scrum**, Extreme Programming, Feature-Driven Development (FDD).

- **Practices:**
    - Pair programming, test-driven development, stand-ups (daily meetings), sprint planning, and sprints (iterations).

Agile Manifesto



The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, **we value the items on the left more.**

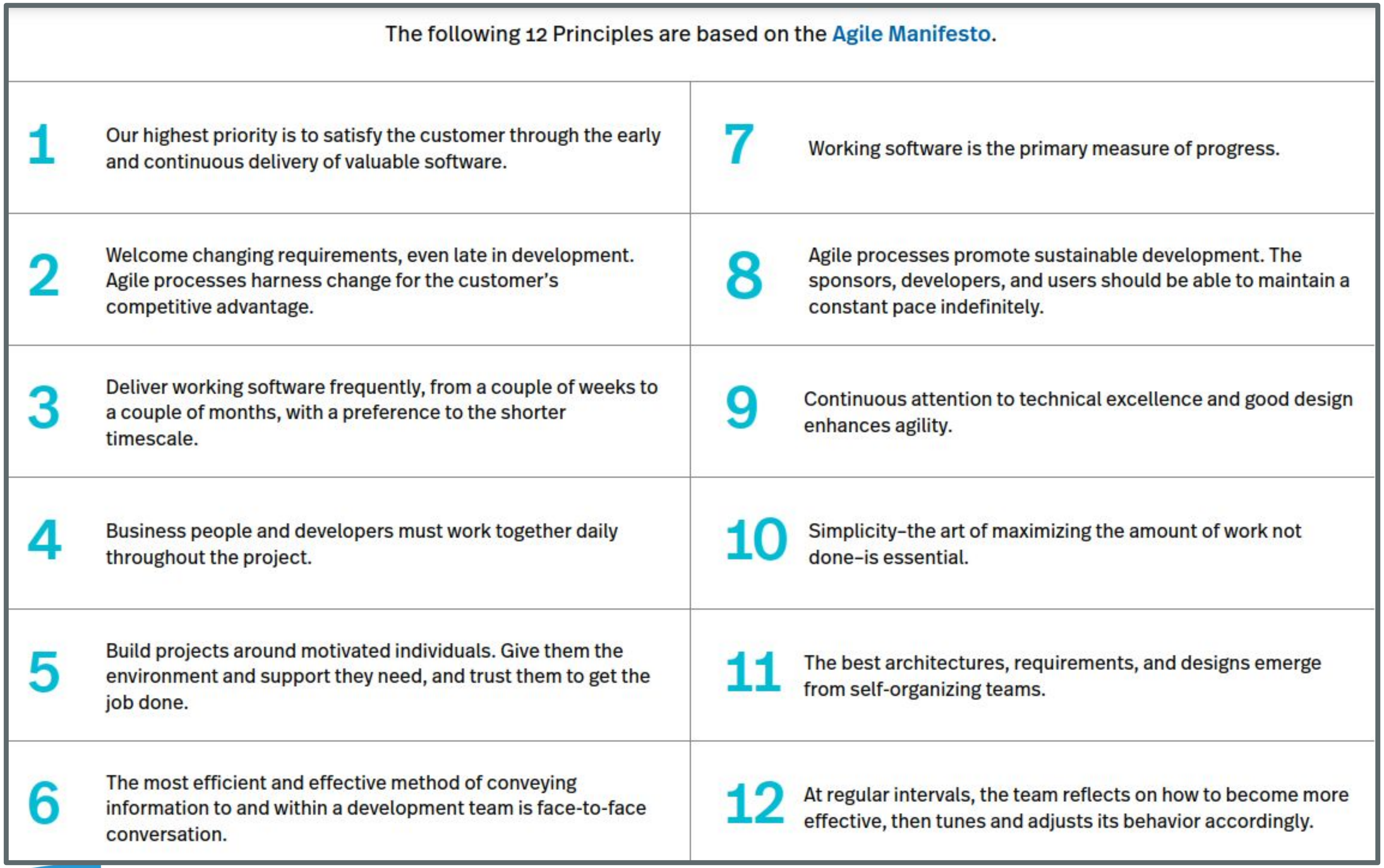
Read the 12 Principles behind the Agile Manifesto.

The Authors

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
Robert C. Martin
Steve Mellor
Dave Thomas
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Ken Schwaber
Jeff Sutherland

Click to learn more about the authors of the Agile Manifesto.

12 Principles Behind The Agile Manifesto



The following 12 Principles are based on the Agile Manifesto.

1	Our highest priority is to satisfy the customer through the early and continuous delivery of valuable software.	7	Working software is the primary measure of progress.
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	9	Continuous attention to technical excellence and good design enhances agility.
4	Business people and developers must work together daily throughout the project.	10	Simplicity–the art of maximizing the amount of work not done–is essential.
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	11	The best architectures, requirements, and designs emerge from self-organizing teams.
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Manifesto in a Nutshell

- Early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development.
- Deliver working software frequently.
- Business people and developers must work together daily throughout the project.
- Face-to-face conversation.
- Self-organising teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Q&A

To follow…

Scrum Software Engineering Framework