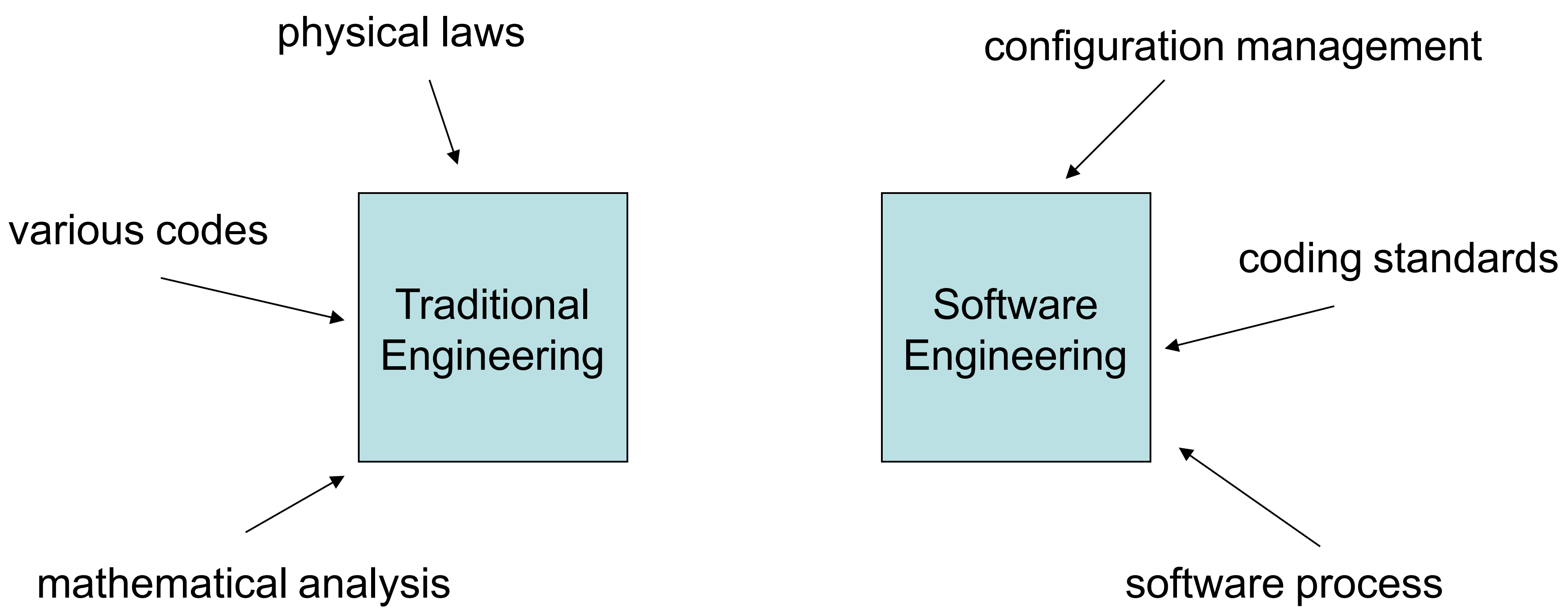
The Minimally Viable Product Process (MVPP) … relating SQA, testing, and management

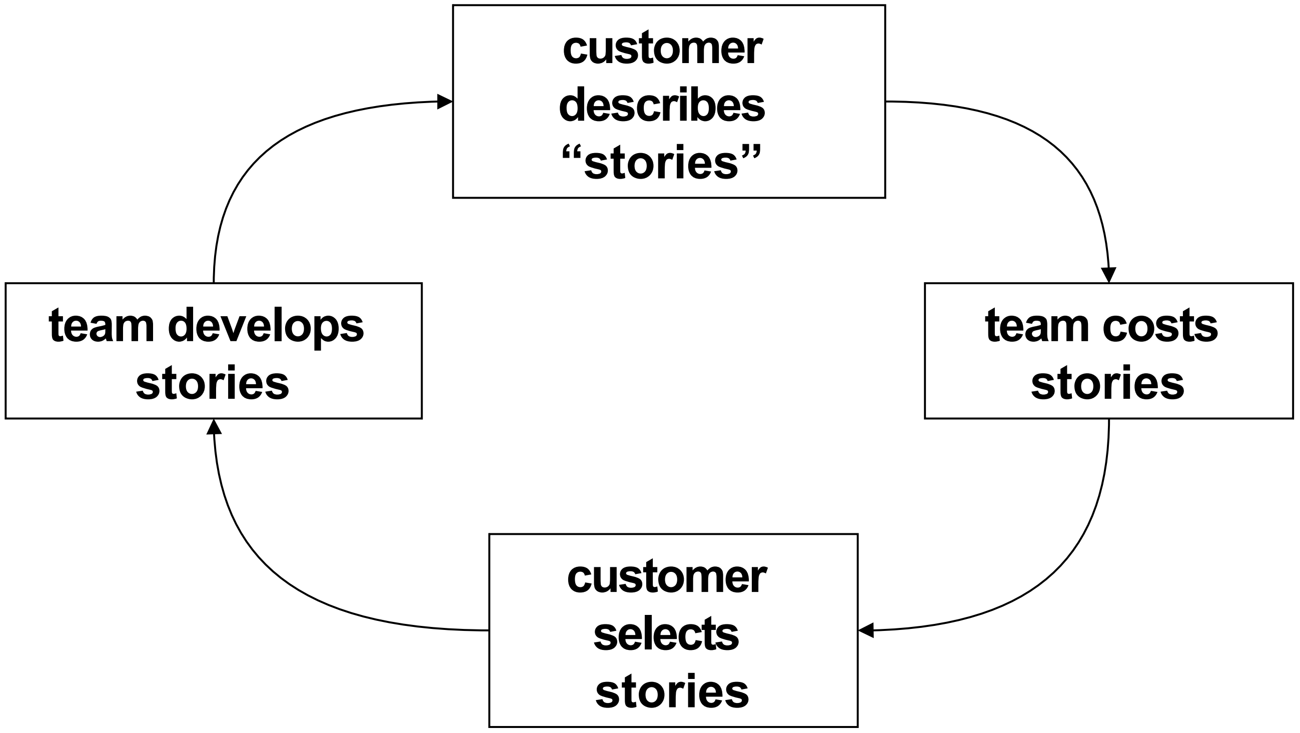
Quality Guarantors



Traditional engineering practices focus on the *product*. Software engineers focus on the process of designing, building and signing a product.

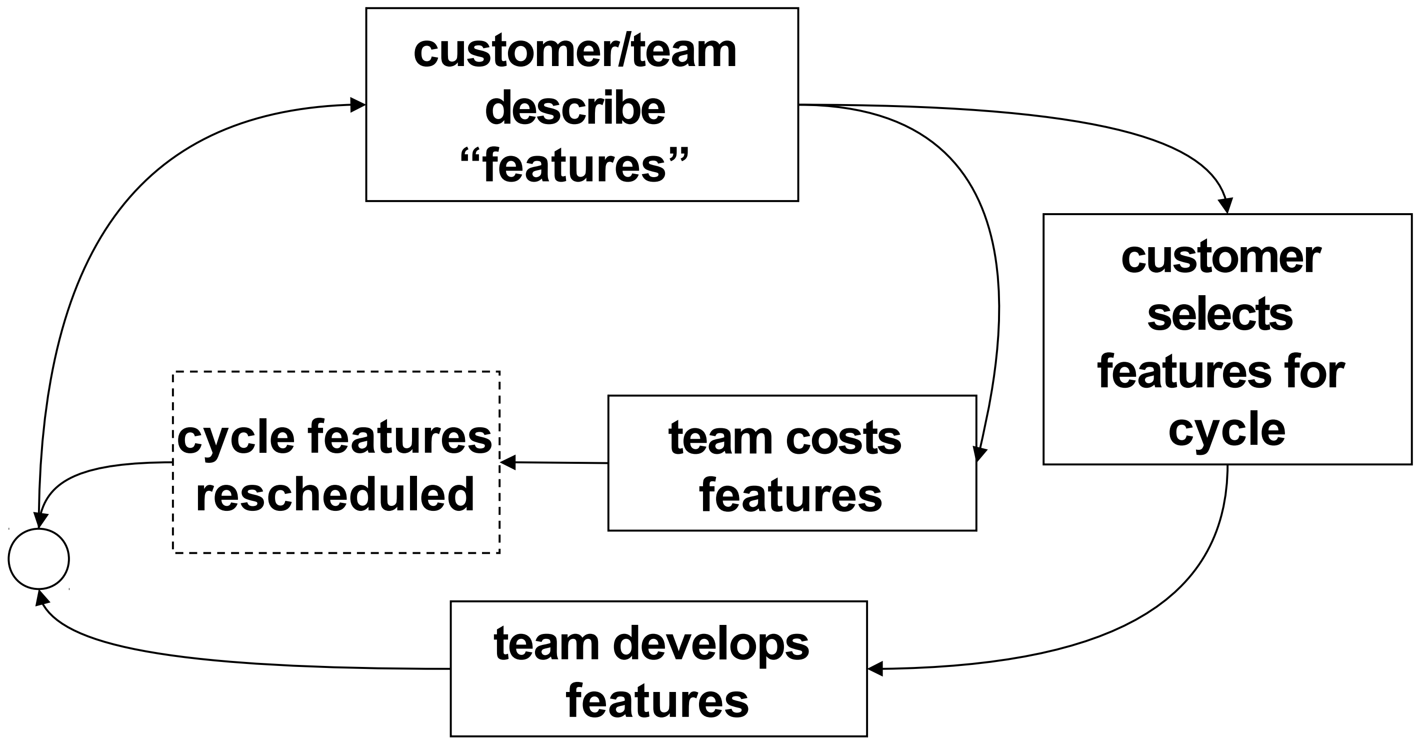
eXtreme Programming (XP)

XP is an "agile" process: focus is on quality delivered via software products rather than through documentation



Minimum Viable Product Process (MVPP)

MVPP is an agile process container: focus is on quality delivered via minimally viable software and demonstrated by minimally viable documentation


customer/team describe "features"
customer selects features for cycle
team costs features
cycle features rescheduled
team develops features

Process Drivers

System Intent

- Described by Customer
- Written by Team

User features

- Described by Customer/Team
- Selected by Customer
- Cost by Team
- Implemented by Team

Cycle Intent

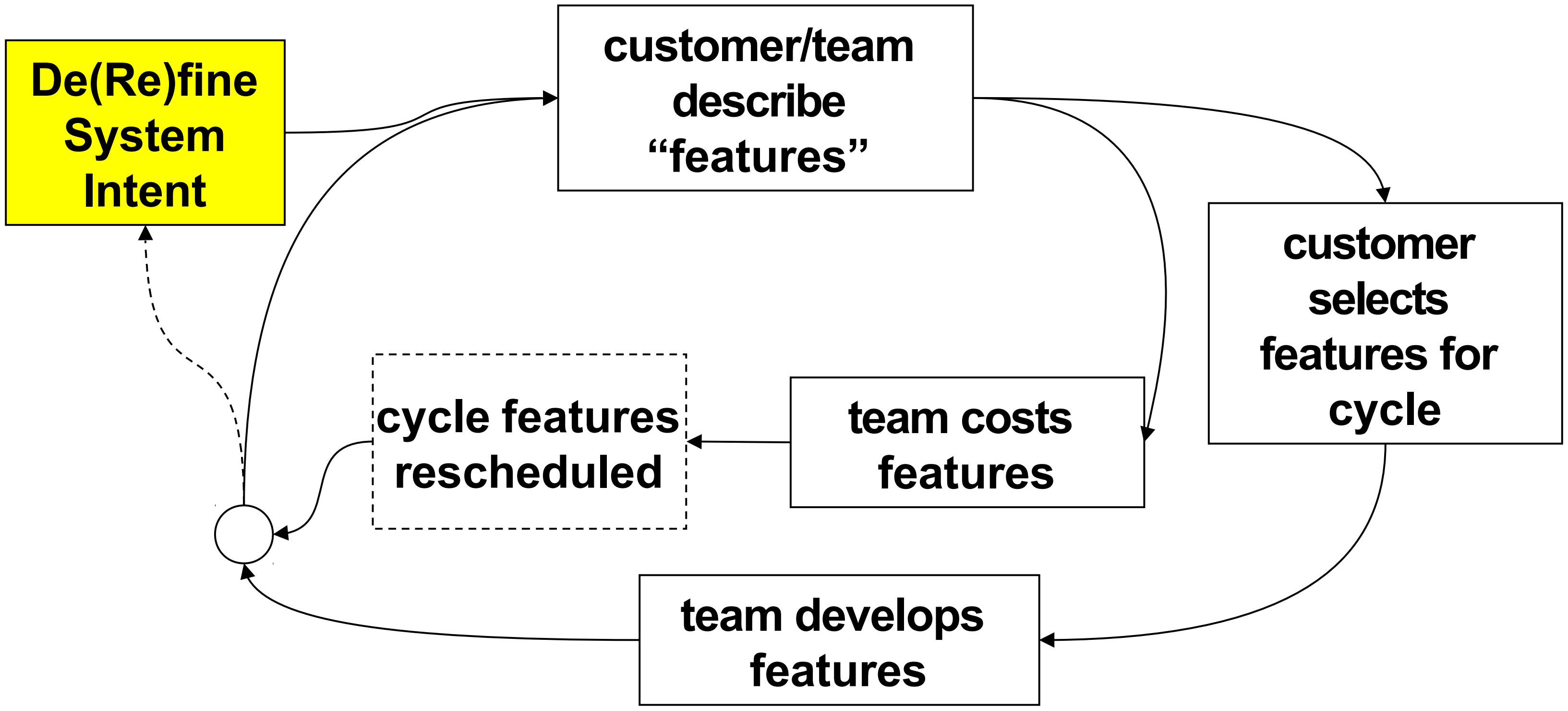
- Described by Customer/Team
- Written by Team

System Intent

The “System Intent” is the mission statement and focus of the team.

“To profitably provide good pizza fast and cheap” [2002]

“To profitably provide consistently good food and great service!” [2004]


De(Re)fine System Intent
customer/team describe “features”
customer selects features for cycle
team costs features
cycle features rescheduled
team develops features

Process Drivers

System Intent

- Described by Customer
- Written by Team

User Features

- Described by Customer/Team
- Selected by Customer
- Cost by Team
- Implemented by Team

Cycle Intent

- Described by Customer/Team
- Written by Team

MVPP Feature

Name: Off-screen ghost markers

Feature: The player must have some awareness of where each ghost is on the playfield as well as whether the ghost is in “kill” or “run away” mode. Because only a portion of the playfield is visible at a time, some sort of marker system should be used to allow the player to keep track of ghosts that are off screen.

Constraints: The screen real estate available to the application should not decrease as result of the marker system.

Note: Features should be detailed enough to estimate the time required to implement them, but they should stay away from low level details. Watch out for BIG features (break them up into subfeatures. Watch out for TINY features… the documentation should be a means to the quality product, not a byproduct.

Process Drivers

System Intent

- Described by Customer
- Written by Team

User Features

- Described by Customer/Team
- Selected by Customer
- Cost by Team
- Implemented by Team

Cycle Intent

- Described by Customer/Team
- Written by Team

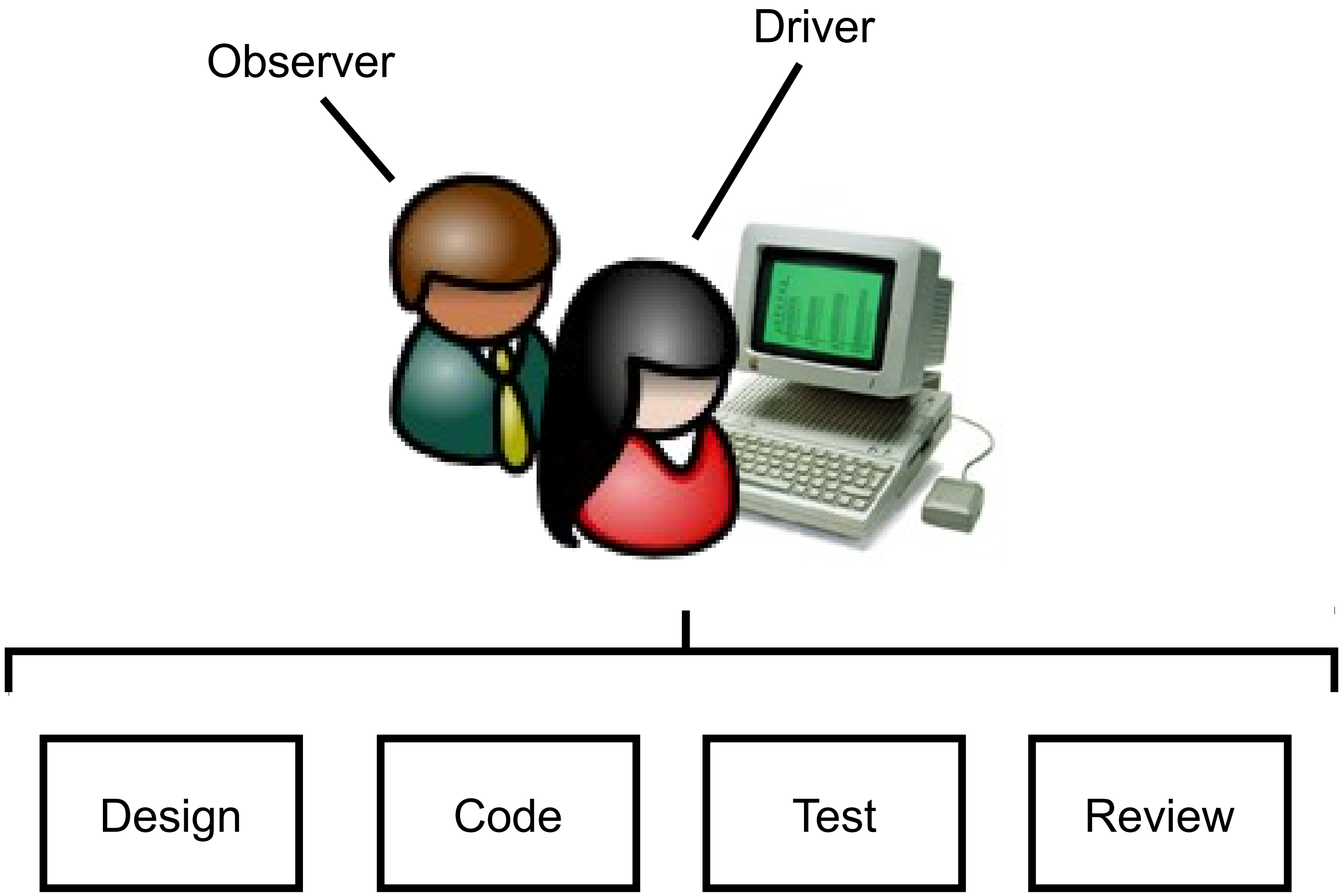
Cycle Intent (a Cycle 1 example)

This cycle will deliver an MVP, particularly splash screen, board layout, ghost and pacman *play* animations, and lives. Basically:

- The board will show in in portions (w/o fancy animations from portion to portion)
- Ghosts will choose a random direction to travel when they hit an intersection
- Ghosts will be blue in “run away” mode and instantly regenerate in the center when eaten
- You can navigate pacman for three lives

Additionally we would like to accentuate the board aesthetics (e.g., scrolling) and improve ghost animations (possibly flashing prior to returning to “kill” mode, having animated dead-ghost eyes, better turn decision algorithm, etc).

Pair Programming



Pair Programming Effects

- **Continuous Review**
    - mistakes caught as they are typed
- **Pair Relaying**
    - two heads are better than one
- **Pair Pressure**
    - positive “don’t let me down” pressure
- **Pair Learning**
    - two heads learn faster than one
- **Satisfaction**
    - Pair experience more enjoyable than working alone
- **Design Quality**
    - Pairs tend to produce shorter programs

Pair Programming Rubs

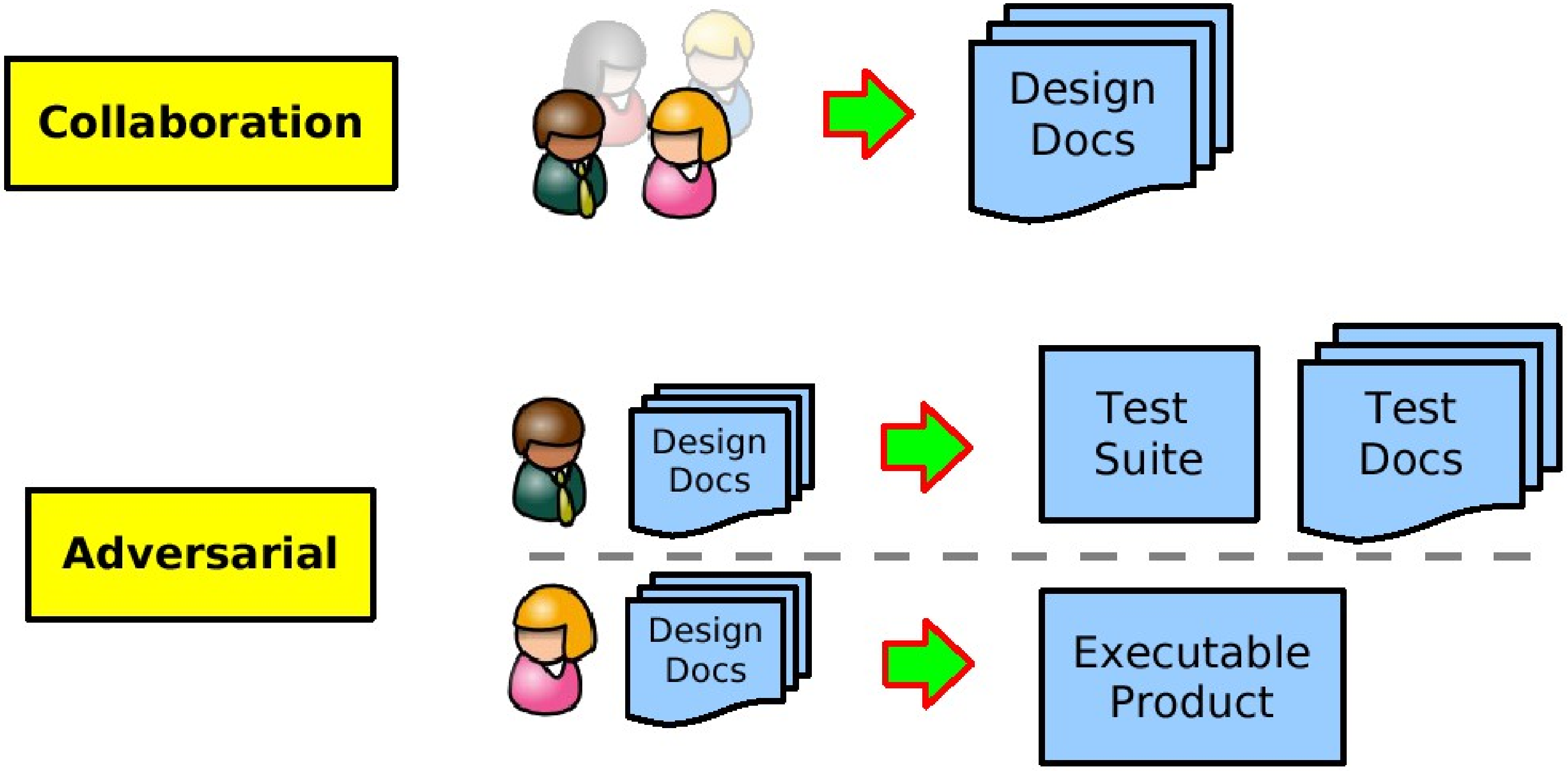
Co-location

Efficiency

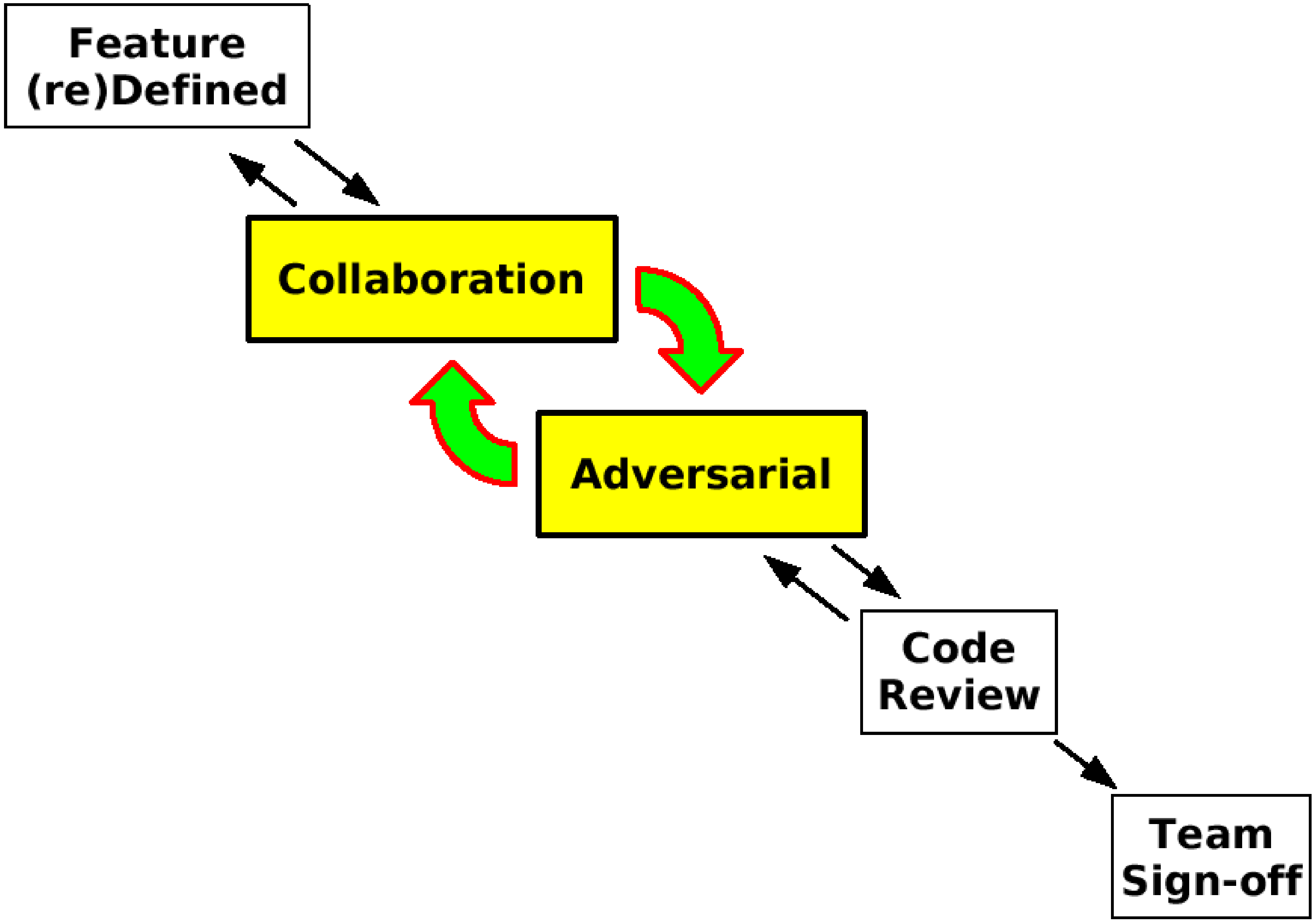
Accountability

What you're saying is that you physically watch the other developer type?

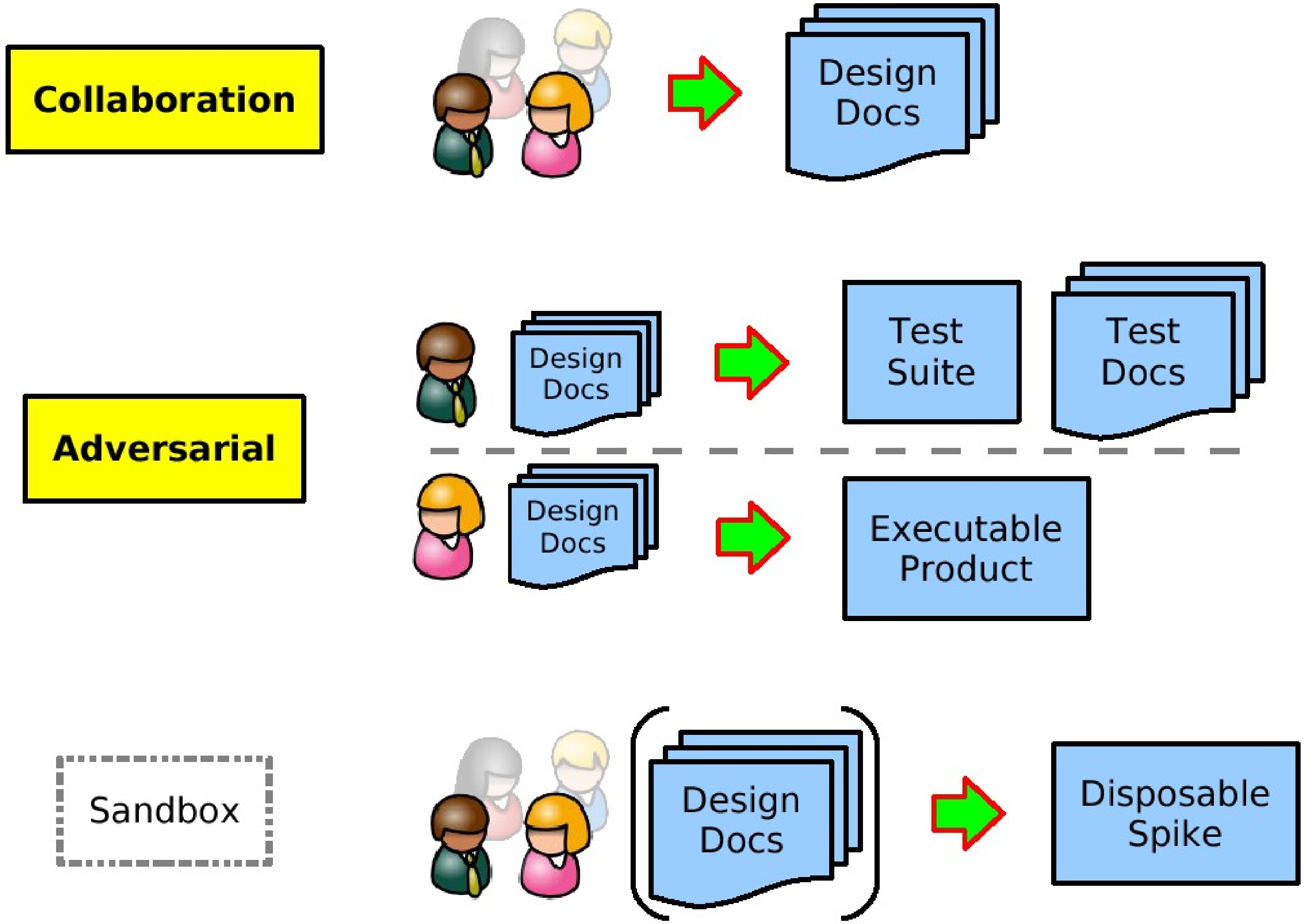
Collaborative Adversarial Pairs



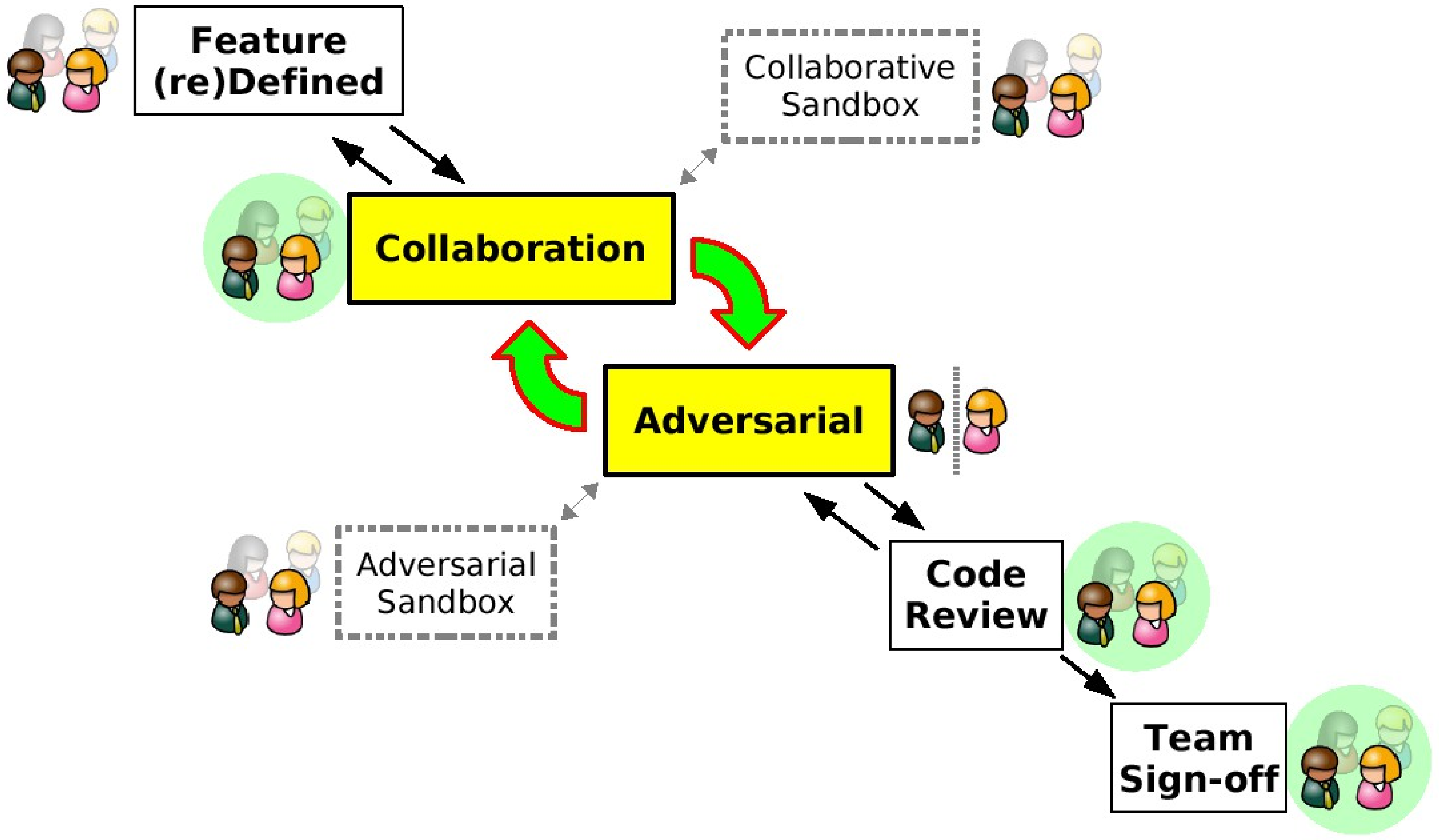
Collaborative Adversarial Pairs



Collaborative Adversarial Pairs



Collaborative Adversarial Pairs



CAP Pros and Cons

Pros:

- Pair relaying
- Pair pressure
- Pair learning
- Satisfaction
- Design quality

- Minor co-location requirements
- More efficient depending on feature size

Cons:

- No continuous review

Deliverables

Hard Copy:

- **Administrative Digest**
  - Management schedules
  - Formal presentations
  - Status reports
  - Memoranda
  - Project evaluations
  - Lessons learned
- **Software Development Folder**
  - User features (w/ prose)
  - Design (w/ prose)
  - Test material (w/ prose as needed)
  - CAP time logs (only if requested)
  - Source code
  - Version description

Soft Copy:

- **Cycle CD** (turn in with the Software Development Folder)
  - Electronic version of all the above
  - Installation-ready build