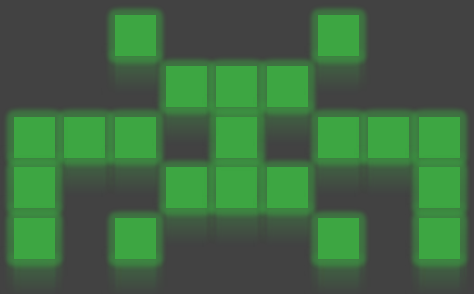
FIND THE BUG! AGILE

A Game of Test-driven Development in an Agile IT Project



for 3-6 players
(playing time 30-60 min)

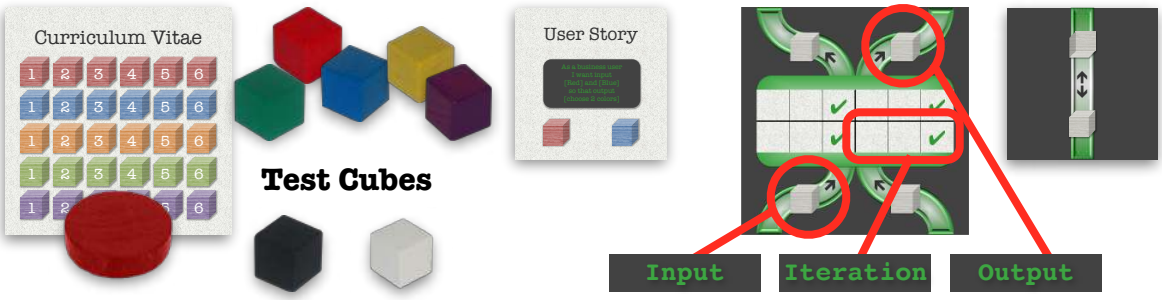
by Nicholas Hjelmberg
Nova Suecia Games
http://www.novasuecia.se
Version 1.0

Internal

The project has been approved by the steering committee and you have been assigned as a scrum master. Please proceed with the sprint planning. Your first task is to identify resources and goals.

1. Form the team: Take one CV and one checkout marker each.
2. Identify the sprint goal: Agree on how many components to develop or integrate. The more components, the longer the sprint.
3. Prepare the test tool: Place the test cubes (black and white) in a pile.
4. Prepare the code repository: Take out code cubes (colored) equal to Player count*2 + Component count of each color (e.g. 4 player and 6 components = 14 codes). Draw six random codes each and keep them open. Place the remaining codes in piles sorted by color.
5. Prepare the user stories: Shuffle the user stories. Draw two random user stories each and keep them hidden. Put aside the rest.
6. Prepare the environment: Put the components and stubs aside. Each component has two input slots, four pairs of iteration slots and two output slots where you may place codes during the sprint.

CV+Checkout **Code Cubes** **User Story** **Component** **Stub**



You are now ready to start the sprint.

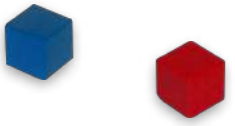
Internal

Is the sprint planning completed? Good, then it's time for the first scrum meeting! Your goal is to use codes to model, develop and test components. This will let you place codes on your CV and the more codes of each color you have at the end of the sprint, the better you performed.

Starting with the most junior scrum master and continuing clockwise, you will now take turns to choose one (and only one) of the following tasks:

1. Pair programming (page 3)
2. Story-driven modeling (page 3)
3. Iterative development (page 4)

Internal



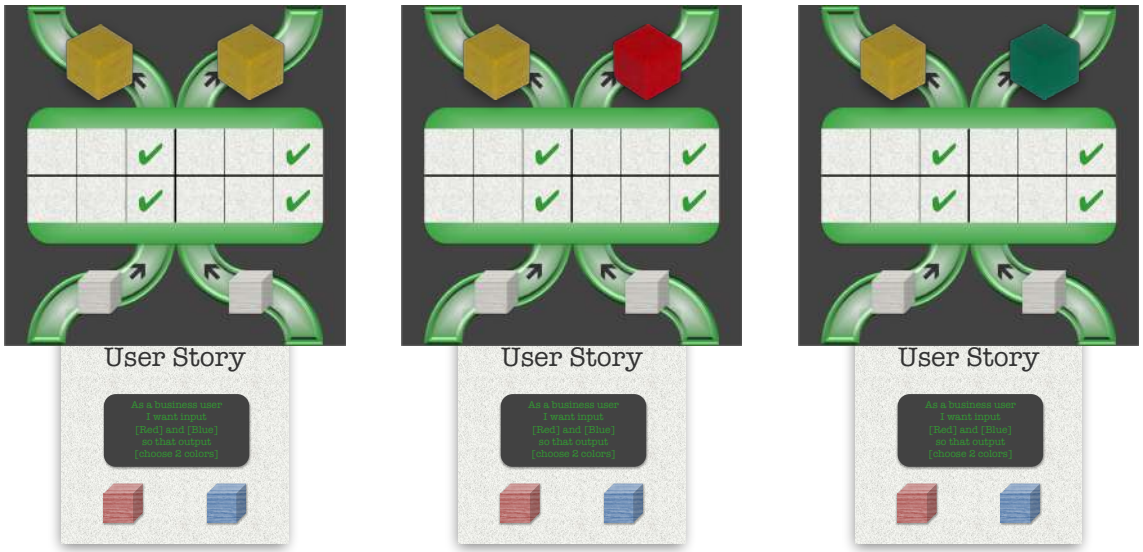
Take one (and only one) of the following code actions:

1. Write: Take one code of your choice from the repository to your hand.
2. Copy: Return all your codes from your hand to the repository. Then take codes similar to those held by another scrum master of your choice from the repository to your hand (but no more than one of each color).
3. Share: Return one pair of equally colored codes from your hand; one to the repository and one to your CV. If you already have a code of a color on your CV, advance it one step instead.

Internal

1. Take one component of your choice and place it in front of you.
2. Choose one of your user stories, place it under the component face down, and draw a new one (if available). It will tell you the required input.
3. Finally choose two of your codes and place them on top of the component, one in each output corner. They will tell the others the expected output. The output codes must be of two different colors and they must be of different colors than the input codes.

You are now product owner for a component that only others may test. If untested, you may return it at any time and get your codes back.



✗ Yellow is used for both output codes.

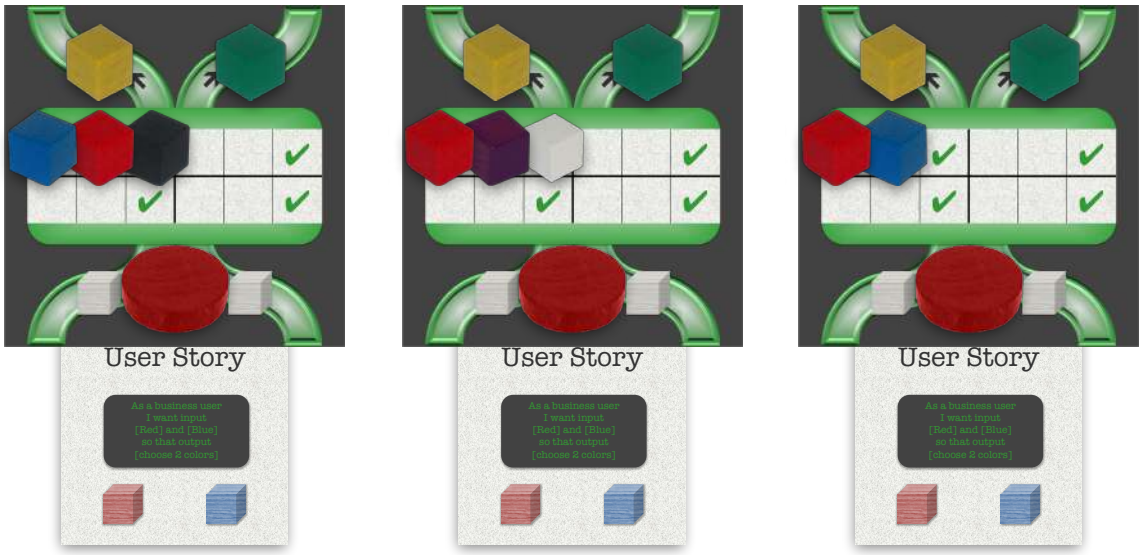
✗ Red is used both as input and output

✓ Input and output are different

Internal

1. Take one component of your choice and place your checkout marker on it. You may not take your own component, one checked out by another scrum master or one already passed (i.e. the input slots must still be empty).
2. Place two differently colored codes of your choice as test input in the first available iteration, one to the left and one to the right.
3. The product owner looks at the user story and compares the iteration input with the required input. To match, both the color and the slot (left or right) must be the same as in the user story.
4. The product owner marks the test result with a test cube.

- If no colors match, place a FAIL marker (black) next to the iteration. The product owner takes two similar codes from the repository.
- If only one color matches, place a PARTIAL PASS marker (white) next to the iteration. The product owner takes two similar codes from the repository.
- If both colors match, move the codes to the input slots. Take two codes of the same color as the output codes from the repository and place on your. CV. The product owner takes no codes.



✗ FAIL - Neither Blue, nor Red match.

✗ PARTIAL PASS - Only Red matches.

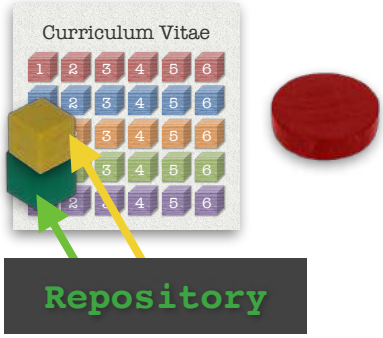
✓ PASS - Both Red and Blue match

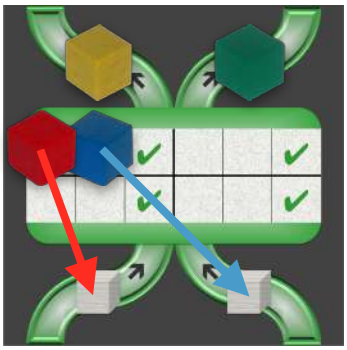
Once a component has passed, return any iteration codes and test result markers to the repository and take back your checkout marker.

You are now technical lead for a component that you or others may integrate. You keep the component and the product owner keeps the user story.

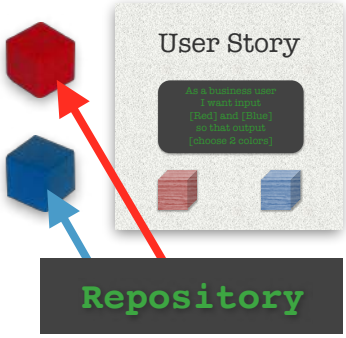
Internal

Tester:





Product Owner:



- ✓ Yellow and Green to CV.
- ✓ Component and checkout to hand
- ✓ Red and Blue to hand.
- ✓ User story to hand.

If the component has not passed after four iterations, the product owner reveals the user story and draws a new one (if any). The component and all codes on it are returned to the repository.

If it turns out that a component had an invalid combination of codes, place two codes on your CV as if the test had passed AND take back all your codes from the component. The product owner is disqualified.

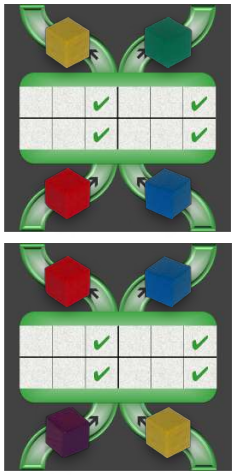
Internal

Are you ready for more challenges? If no, go to the Retrospective when the agreed number of components have been developed. If yes, continue with the Continuous integration until the agreed number of components have been not only developed but also integrated.

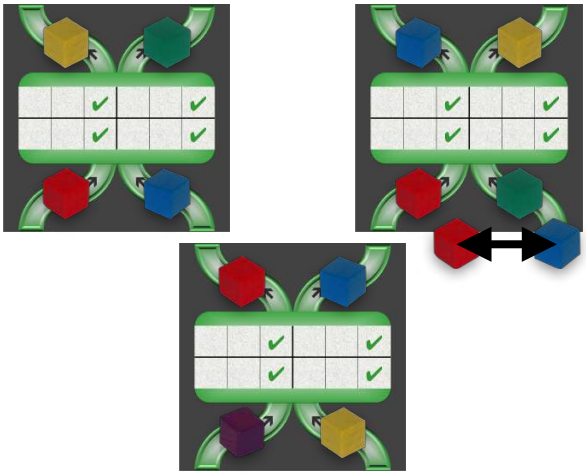
Besides the three tasks above, you may choose a fourth: Continuous integration of components into one module.

1. Take as many passed components and stubs as you want to integrate. You must integrate at least one new component and you must integrate new components to an existing module if any.
2. Arrange the components in distorted columns so that they are linked by input and output slots. The slots must be of the same color. You may switch left and right colors.
3. You may add stubs to complete ”missing links” between two linked components. The slots may be of any colors.
4. For each ”open” input slot, return codes of that color from your hand to the repository. You must have all the required codes to integrate.

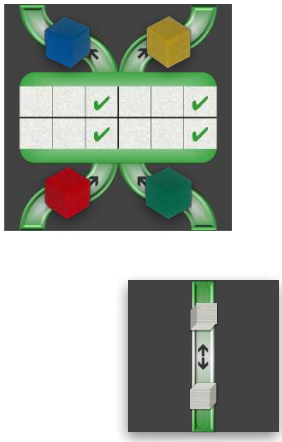
Internal



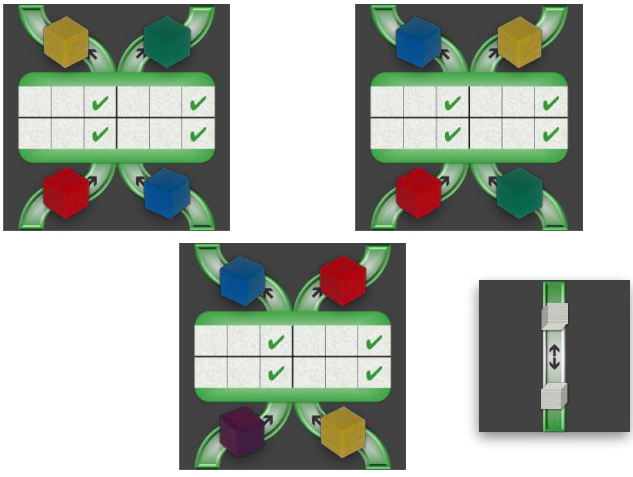
✗ The components are not distorted.



✗ The slots are not of the same colors (but red and blue can be switched to match)



✗ The stub cannot be the only link between components.



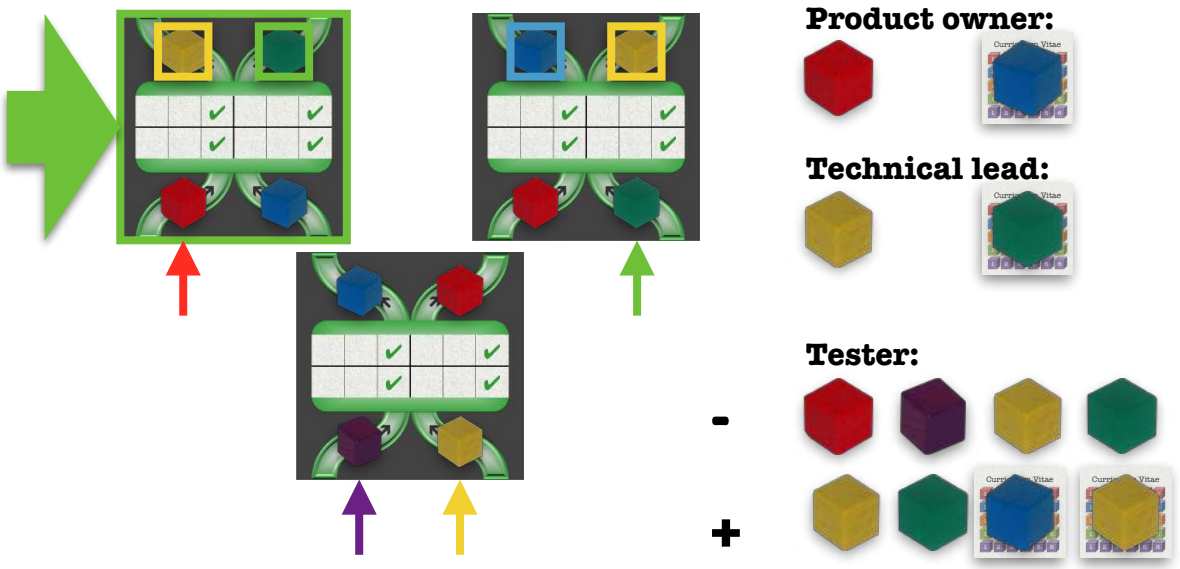
✓ The stub can be used to link an output of one color (blue) with an input of another color (green).

Internal

Once integrated, remove the stubs but keep the components integrated. They may not be separated again.

A successful integration rewards the following from the repository (in clockwise order, starting with scrum master in turn):

1. The product owner of the newly integrated component (the player with the user story): The component input codes, of which one of each component is placed on the CV.
2. The technical lead of the newly integrated component (the player with the component): The component output codes, of which one of each component is placed on the CV.
3. The tester (the player who integrated the components): The module output codes, of which half are placed on the CV (rounded down).



Credits

Game design and art: Nicholas Hjelmberg
Production: The Game Crafter
Game testers: Juan Garcia, Hans Larsson, Eva Nordström, Lotta Östergren, colleagues at SQS Sweden
Special thanks: My wife Su-San Oh for her patience, my SQS colleague Hans Larsson for inspiration in the test area and my game colleague Nerdfest Games for advice on how to bring testing to a board game

Internal

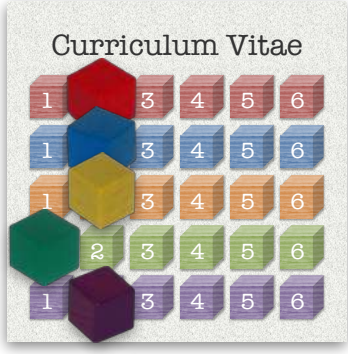
The end of the sprint is triggered at the end of a scrum master's turn if:

- the agreed number of components have been passed/integrated or
- there are not enough codes left of a color to give to a scrum master

All scrum masters get one more turn, after which the sprint ends. Use codes from outside the game as necessary so that all get their codes.

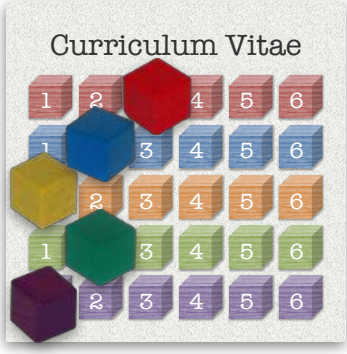
It is now time to look back on what you have achieved. Look at your CV and report the number of codes of your smallest color. The scrum master with the highest number wins. In case of a tie, report your second smallest color and so on. If the tie remains, report the total number of codes left in your hand.

Srum master 1



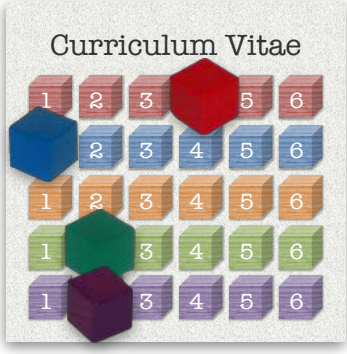
Smallest: 1 (Green)
2nd smallest: 2 (Red)

Scrum master 2



Smallest: 1 (Yellow)
2nd smallest: 1 (Purple)

Scrum master 3



Smallest: 0 (Yellow)

You may now close the sprint. Please remember to continuously improve your way of working. Could you have spent less or more time on any task? Did you test too little or too much in any component? Do discuss your strategies with your colleagues and learn from each other.

We thank you for your effort and look forward to working with you again.

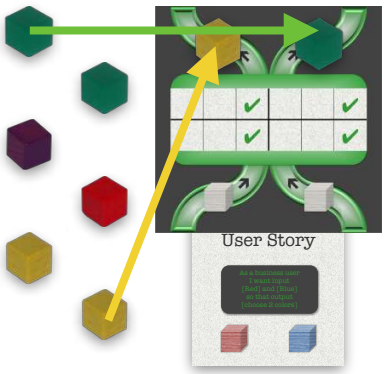
Game Components

12 component tiles
6 CV tiles
6 epic tiles (rules)
20 user story cards
4 stub cards

6 checkout markers
120 code cubes; 24 red, 24 blue, 24 yellow, 24 green, 24 purple
12 test cubes; 6 black, 6 white

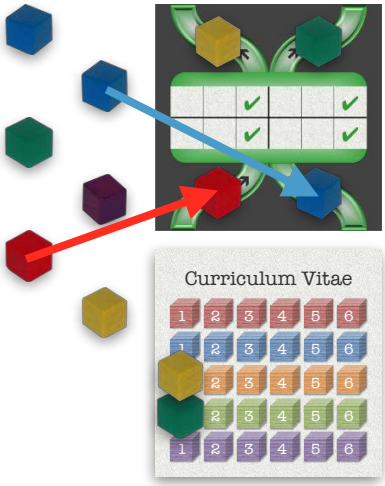
Sample Round

Player 1 has Green, Green, Purple, Red, Yellow and Yellow codes on the hand. She chooses the Story-driven modeling tasks to take one empty component and place one of her two user stories beneath it. Since it has input Red and Blue, she must choose different output and places Yellow and Green on the output slots. Finally she draws a new user story and has two user stories again.



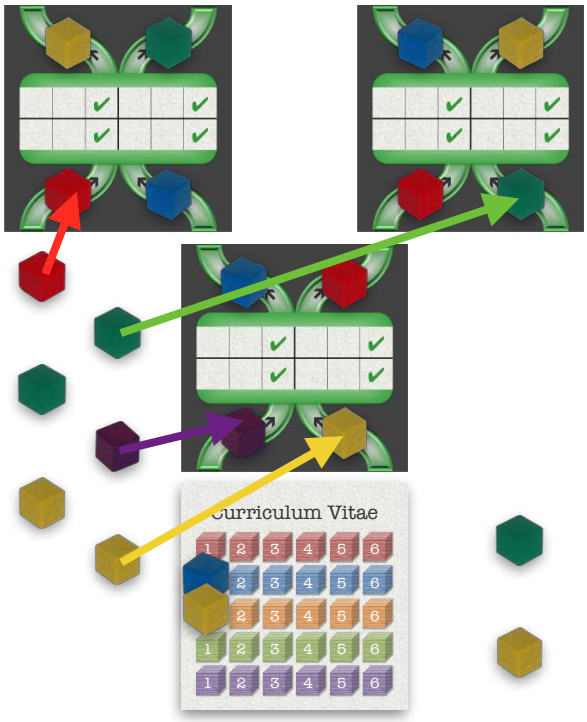
Player 2 has Blue, Blue, Green, Purple, Red and Yellow codes on the hand. She chooses the task Iterative development to place Red and Blue on player 1's component together with her checkout marker. Player 1 confirms that this is the combination on the user story so the test passes.

Player 2 takes back her checkout marker and moves the codes to the input slots. Player 2 now gets Yellow and Green codes from the repository to place on her CV. Player 1 consoles herself with Red and Blue codes from the repository to her hand.



Player 3 has Green, Green, Purple, Red, Yellow and Yellow. Previous round, a module was integrated linking red input and output. Player 3 now integrates the component passed by player 2 by linking the blue output from the module with blue input from the component.

To do this, she needs to place Red, Purple, Yellow and Green input codes in the repository. In return, she chooses to place the module output codes Yellow and Green in her hand and Blue and Yellow on her CV. Player 1 gets the component input codes Red and Blue (one to the CV). Player 2 gets the component output codes Yellow and Green (one to the CV).



Player 4 has only Purple and Purple left on her hand. She chooses the task Pair Programming to copy player 1's code. She returns her codes to the repository and, since player 1 has codes of all colors, she gets to take one code of each color (but only one). It is now player 1's turn again...

GLOSSARY

Term	Agile concept	Game concept
Agile development	Requirements and solutions evolve through collaborating between self-organizing cross-functional teams	The players alternate between developer tasks and tester tasks
Continuous integration	Merge of all developer working copies to a shared mainline several times a day	Action whereby one player (tester) places components next to each other so that input and output match
Epic	Collections of related user stories	Rule tiles (or the ”epics of the game”)
Iterative development	Develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental)	Action whereby one player (tester) plays a combination of codes and another player gives the result (product owner)
Pair programming	Two programmers work as pair, one writes code and the other reviews it	Action whereby a player takes codes similar to the codes of another player
Product owner	Responsible for representing the business in the development activity	The player creating a component with a hidden input and an open output
Repository	On-disk data structure which stores metadata for a set of files as well as a history of changes	Stock of code cubes, from which the players may replenish their hands
Retrospective	Held at the end of every iteration to look for ways to improve the process	End game phase whereby the players count their victory points and determine the winner
Scrum master	Facilitator of day-to-day operations of agile teams	Role of players in the game
Scrum meeting	Daily meeting where members tell what they did, what they will do and what impediments they see	Four actions or tasks available to the players each turn
Scrum of scrums	A technique to scale Scrum up to large groups (over a dozen people), consisting of dividing the groups into Agile teams of 5-10	The honor given to the winner (”scrum of scrums master”)
Sprint planning	Identify work and goal for sprint	Setup phase when the game is prepared and game end condition agreed
Story-driven modeling	Focus on concrete example scenarios, on how they may be represented as object diagrams, and on how they may evolve during scenario execution	Action whereby one player (product owner) creates a test case for the other players to test
Stub	A piece of code used to stand in for some other programming functionality	A card used to link output and input instead of a component
Technical lead	Software engineer in charge of one or more software projects	The player passing a component
Test-driven development	Write test case first, then produce code to pass test, and finally refactor code to acceptable standards	Game mechanism whereby the players need to find the right combination of codes to get victory points
Test lead	Test professional involved in the planning, monitoring, and control of the testing activities and tasks	Responsibility of players in the game (i.e. what to test and when)
User story	Description of what a user needs as part of the job function, typically in the form ”As a... I want... so that...”	Card telling which input that is needed to get a given output