The Abelian Sandpile Game

Implementation by Matthew Heusser matt@xndev.om
Derived from the work of Noah Sussman, used with permission

Based on the work on Per Bak, Chao Tang, and Kurt Wiesenfeld
https://en.wikipedia.org/wiki/Abelian_sandpile_model

Background:

Pour sand out in a uniform amount from a standard distance, and it will form the a pile. The pile will grow and grow until the base is unable to support the weight; then the base will expand. One mathematical formula that simulates this effect is the *Abelian sandpile model*.

Software can be remarkably like an abelian sandpile, in that changes to the system add up, introducing small defects, and these defects have a *ripple effect* into other modules. Eventually, the ripple effects create ripple effects, and the system is destabilized.

The Sandpile game simulates this on a 4x4 grid. Ideally this game is played in competition so teams can compare scores, with a single game master who counts the number of

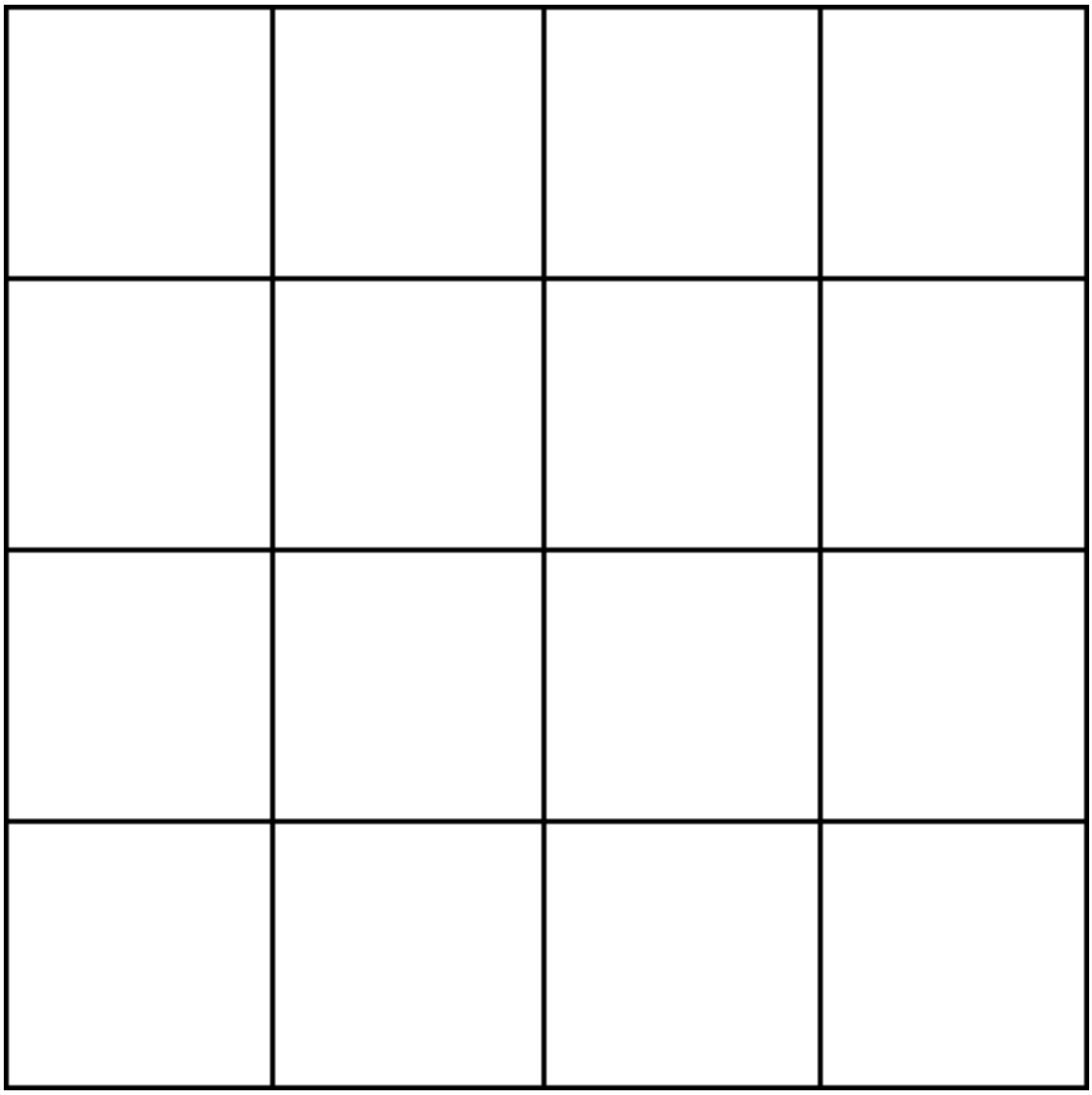
The Rules:

Players select and mark three deploy sites, places to leave code. Mark each deploy site with a large X. Each team gets forty (40) changes to deploy on any of the three deploy sites. Use some sort of counter for each deploy site, such as a USA penny, nickel or dime.

When five counters hit a square, the site has a defect that escapes. To simulate that, break the counters up, with one on each of the squares above, below, to the left, and to the right, and the fifth counter on the original square. *Players of the game pandemic will be familiar with this algorithm.*

Round 1:

The goal is to deploy all thirty changes while minimizing the number of coins that fall off the board.



For the Game Master:

In addition to counting the number of counters that fall off the board, also count the time each team spends getting the thirty counters through. In round 2, play again, but suggest the goal is not to minimize the number that fall off, as much as to get all thirty through the system. Notice how time goes down radically with this change in behaviors.

From there, explain the application to software and specifically DevOps.

This is: Most of software delivery organizations spend a great deal of time trying to reduce Mean Time Between Failures (MTBF) - that means creating a release candidate for our big ball of mud and testing it to death. If we focus instead on creating a resilient system - one where the cost of failure is minimize, by reducing Mean Time To Recovery (MTTR), we can get more done in less time, with less argument.

In software, the typical ways to do this are to create a component architecture, reducing ripple effect, while reducing failure demand and improving production monitoring and rollout speed.

Typical methods for a component architecture include refactoring to APIs, RESTful services, SOFEA for the front-end, separate business logic from data, and so on.

Typical methods for reducing failure demand include code as craft, unit tests, and continuous integration.