
Adopting
Continuous Delivery
Jafar Soltani
Lead Software Engineer
Rare Ltd.
Microsoft Studios

Traditional Development Process

- Monolithic application, developed in C++
- Waterfall Process, Three main phases:
  - Pre-production or prototyping
  - Production
  - Bug fixing
- Heavily rely on an army of testers
- One big release followed by a handful of updates


Sea of Thieves

Sea of thieves

- Multiplayer cooperative adventure game
- Game as a service
- Over 150 releases to technical alpha Audiences
- The game is gone live This week

Why We're adopting continuous delivery

1. Sustainably delivering new features over long period of time
2. Minimising Crunch and having happier developers
3. Getting fast feedback and delivering a better quality game that is more fun
4. Reduce cost of having a large manual test team

Continuous Delivery vs Frequent Release

Frequent Release

Continuous Delivery

Release Release Candidate Broken Candidate

How we're adopting continuous Delivery

How we're adopting continuous Delivery

1 - Developers are responsible for the quality of their feature

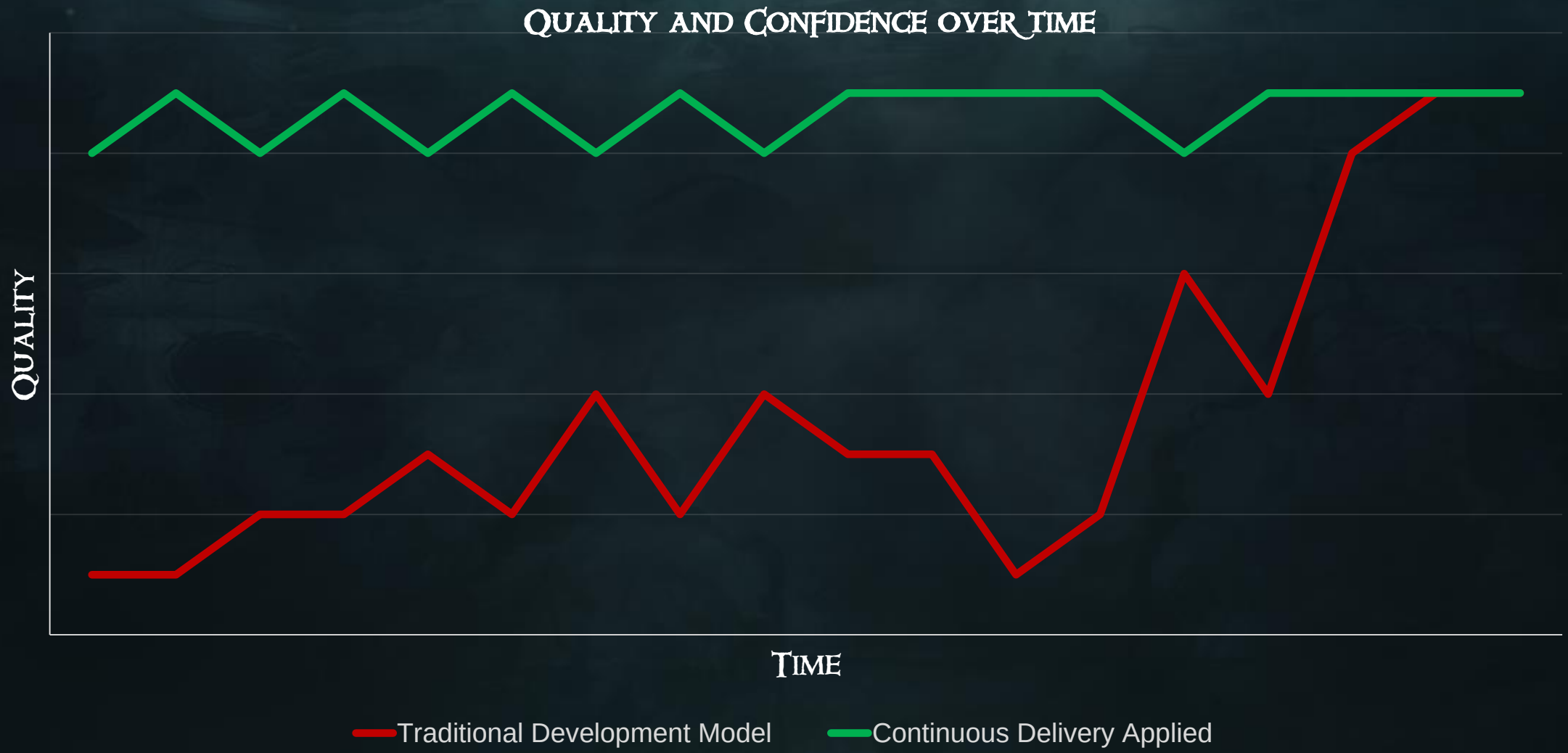
- Very inefficient to verify the game manually
- Developers write automated tests to verify their work.
  - 40,000 automated tests , running 4 million tests each day
  - 90 Percent unit tests in C++
  - 10 Percent End to End, performance and memory tests
- Add regression tests when fixing bugs

How we're adopting continuous Delivery

2 - Game is always shippable

- Prioritise fixing bugs and broken tests over developing new features
- lock The depot every time we can't ship or the commit stage is broken
- Build lights and tv screens to notify everyone

Quality and Reliability of the Game Over Time



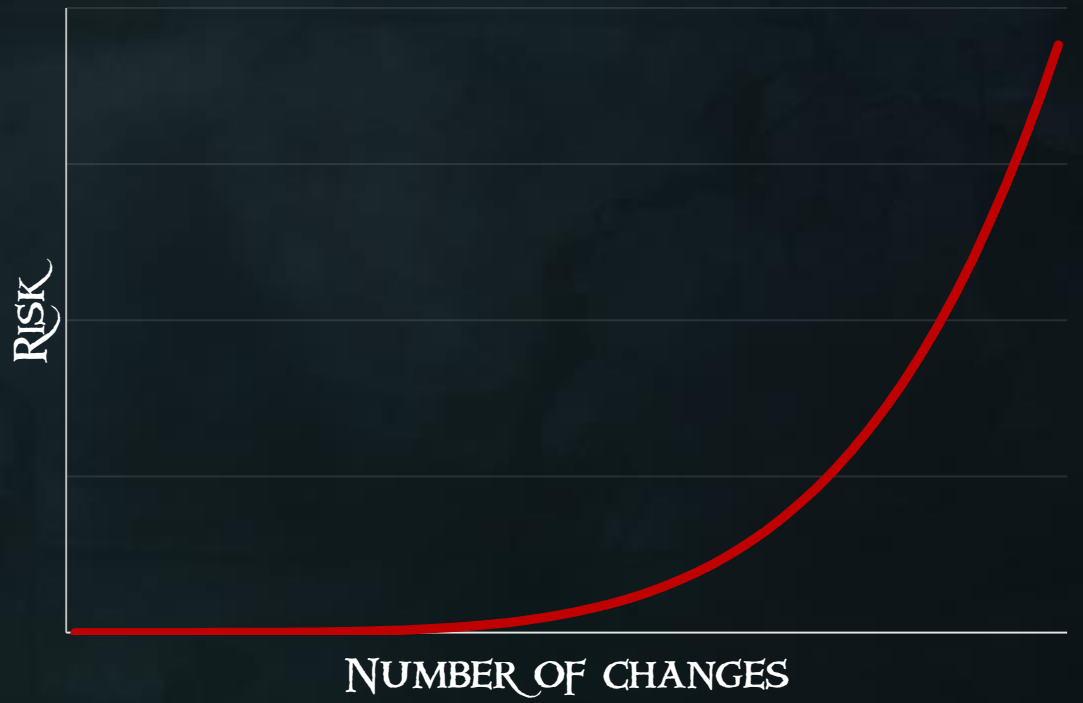
How we're adopting continuous Delivery

3 –Each build contains small number of changes

Developers break down their work into

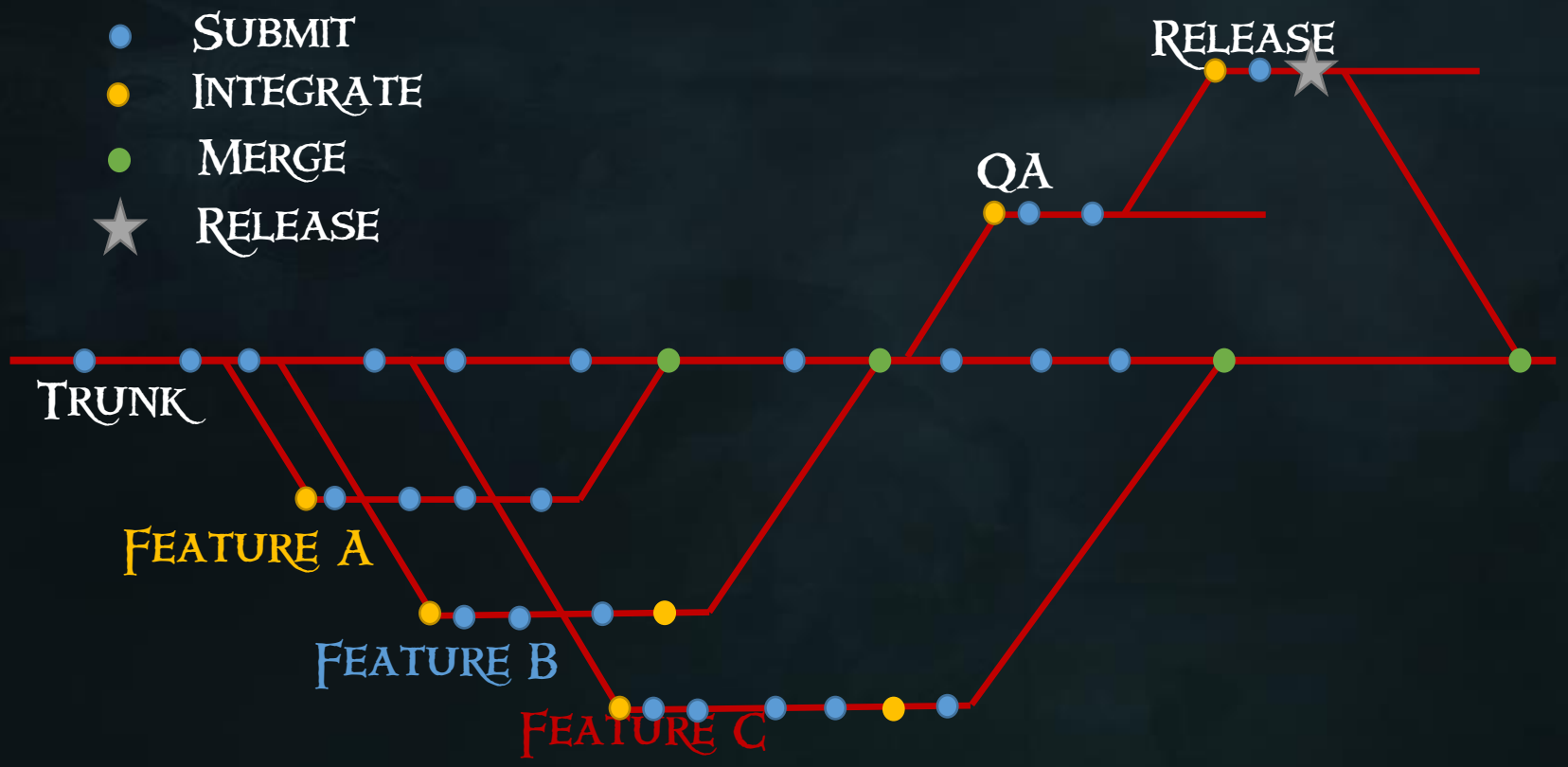
Small chunks and try to checkin once a day

When there are more changes in a build, risk grows exponentially



How we're adopting continuous Delivery

Traditional Branching strategy



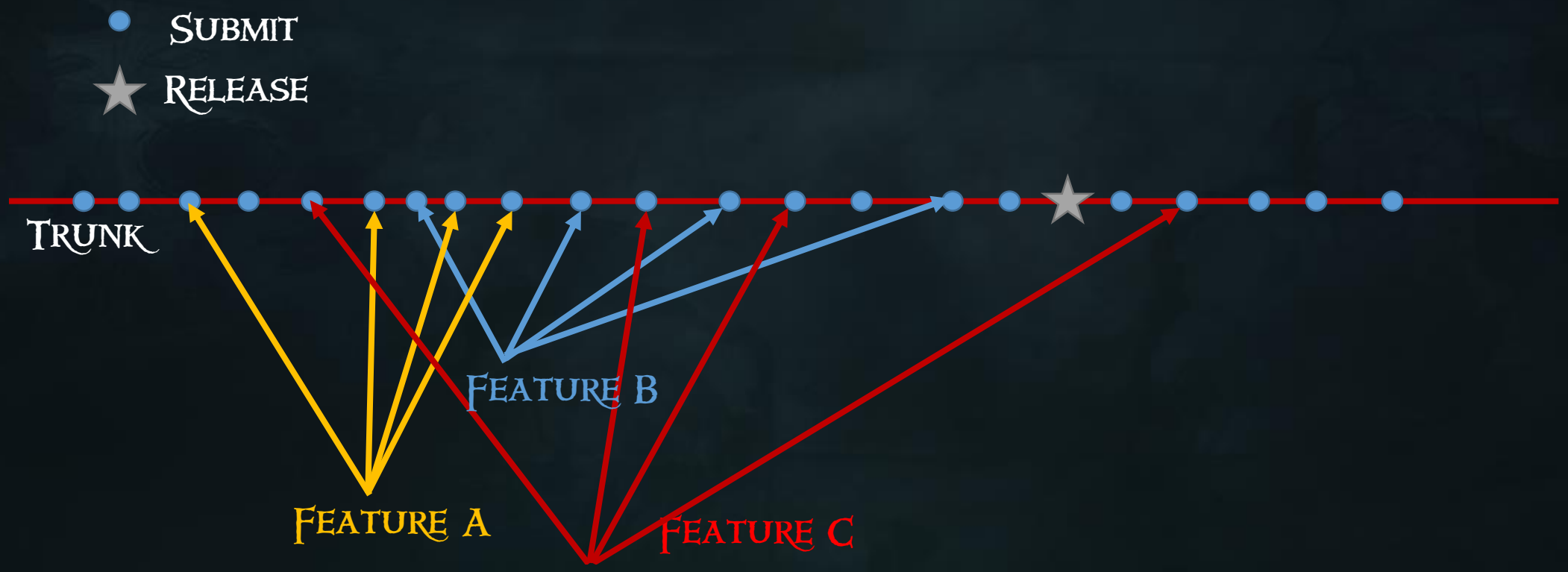
How we're adopting continuous Delivery

Problems with Traditional Branching strategy:

- Painful Merge Conflicts
- Binary files
- Semantic conflicts
- Long feedback loop
- Integrating fixes to multiple branches

How we're adopting continuous Delivery

4 - Trunk-based Development



Feature Toggle

5 - Feature toggles

- Compile-time toggle
  - Allow everyone to work on trunk
  - Prevent in-progress features from being released
- Dynamic toggle
  - Roll out new features to a small set of players, learn and build confidence in the features

Feature Toggle

Compile-Time Toggle

json file

```
{
    "features" :
    {
      "featureA" : {
      "enabled" : true,
      "description" : "This is feature A"
      },

      "featureB" : {
      "enabled" : false,
      "dynamic" : true,
      "description" : "This is feature B"
      }
    }
}
```

cpp file

```
if (GFeatureConfig.IsFeatureEnabled(TEXT("featureA")))
{
    // ...

}
else
{
    // ...

}
```

Feature Toggle

Dynamic Toggle

json file

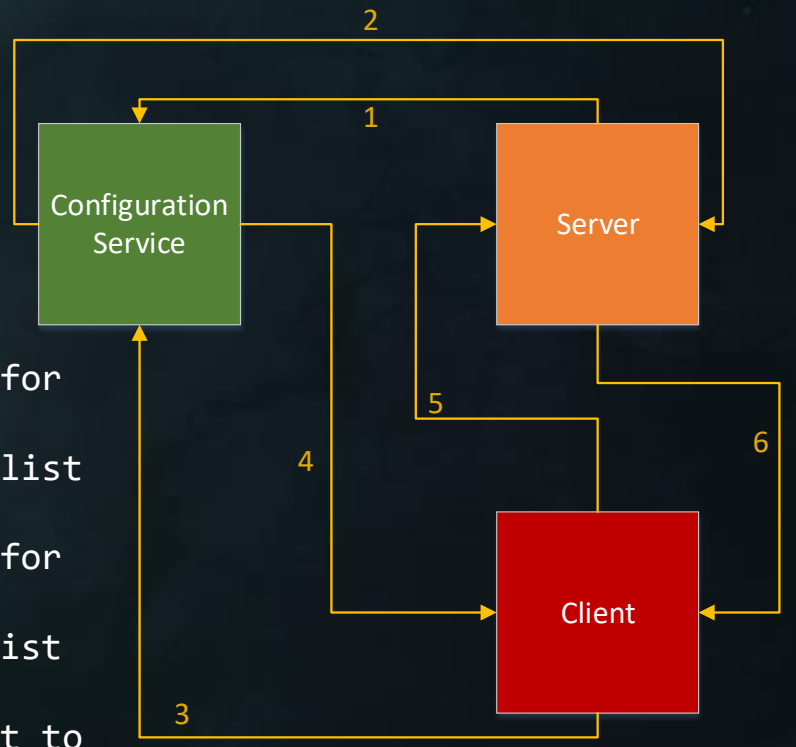
```
{
    "features" :
    {
      "featureA" : {
      "enabled" : false,
      "description" : "This is feature A"
      },

      "featureB" : {
      "enabled" : false,
       "dynamic" : true,
      "description" : "This is feature B"
      }
    }
}
```

1 – Server asks Service for the list of features
2 – Service returns the list

3 – Client asks Service for the list of features
4 – Service return the list

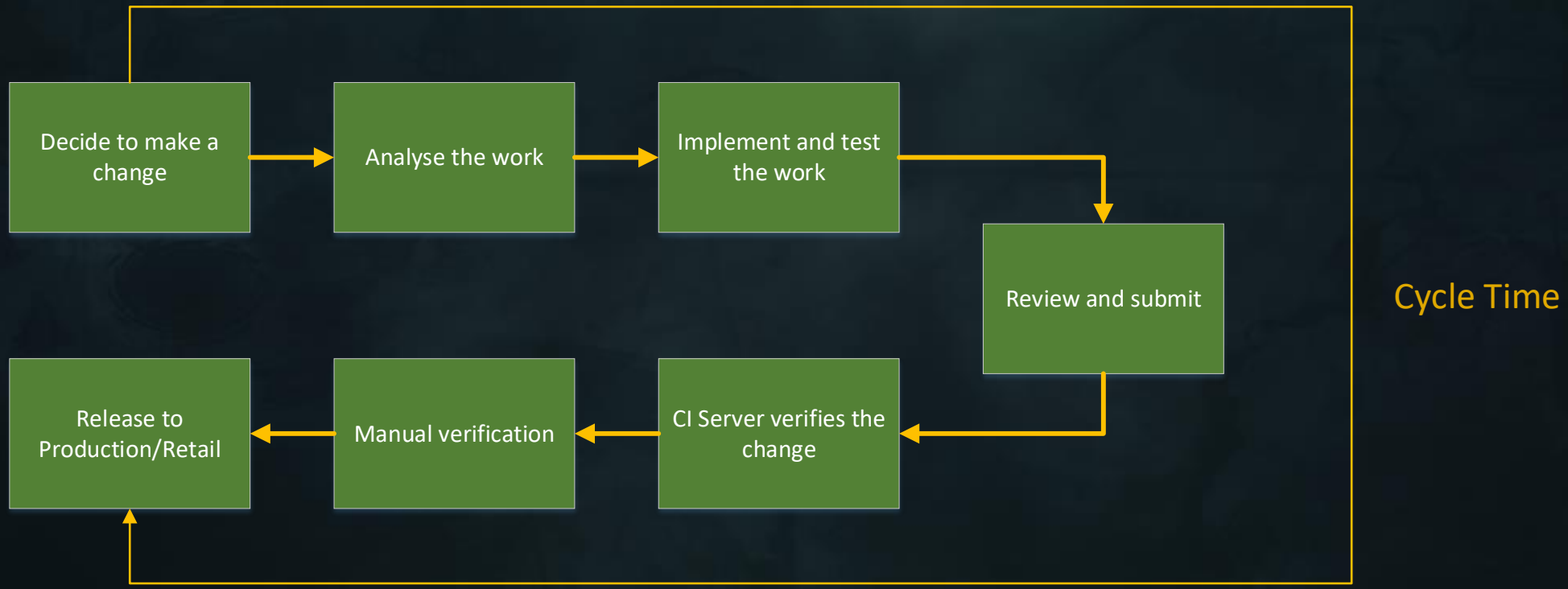
5 – Client sends the list to Server before joining the game
6 – Server matches Client list and either let the Client join or rejects it



How we're adopting continuous Delivery

6 - Continuously improve Cycle Time

What is cycle time



Why we should optimise cycle time

Benefits of short cycle time:

- Fast feedback loop, leads to better quality
- Enables working in small batch and reduces risk
- Reduces hand-overs,, leads to cross-functional teams
- Fast response time

Deployment Pipeline

Deployment Pipeline is the Implementation of our process developing and releasing features to players.

We formed A team of engineers to implement the deployment pipeline

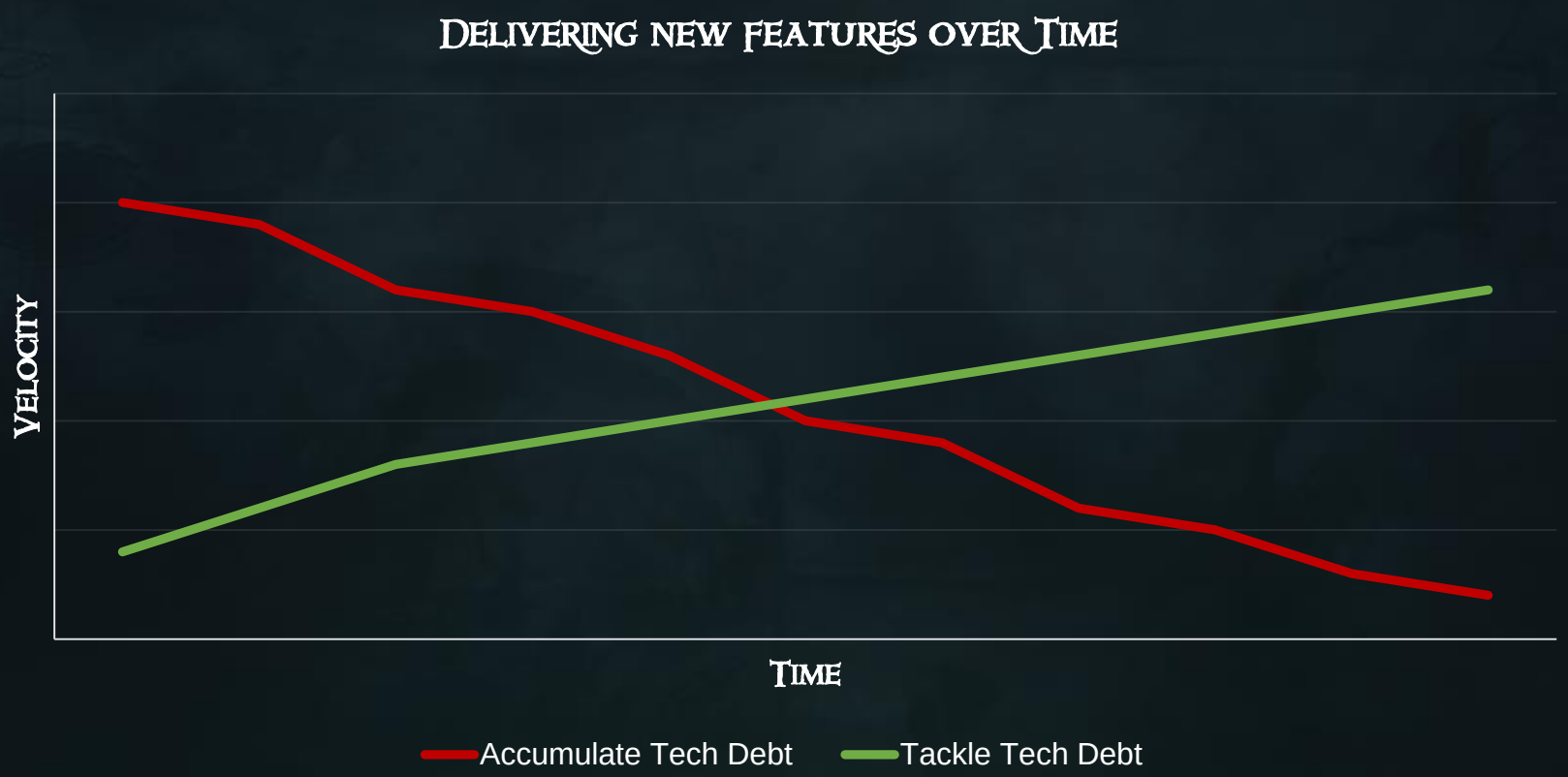
Deployment Pipeline

What Deployment Pipeline team does:

- Develop test framework
- Responsible for the workflow, infrastructure and pipeline
- Improve cycle time:
  - Develop system to identify flaky tests
  - Feature toggle
  - Improve build time, Cook time
  - Parallelise running tests

How we're adopting continuous Delivery

7 - Impact of technical debt on velocity



When to tackle technical debt

Tackling technical debt regularly and improving code maintainability

How we're adopting continuous Delivery

8 - Continuous improvement:

- Our most important principle. We're building a learning organisation
- Regular retrospectives
- Post-mortem after every incident
- Allocate time to complete actions from retrospectives and post-mortem
- Evolved our process and principles over time

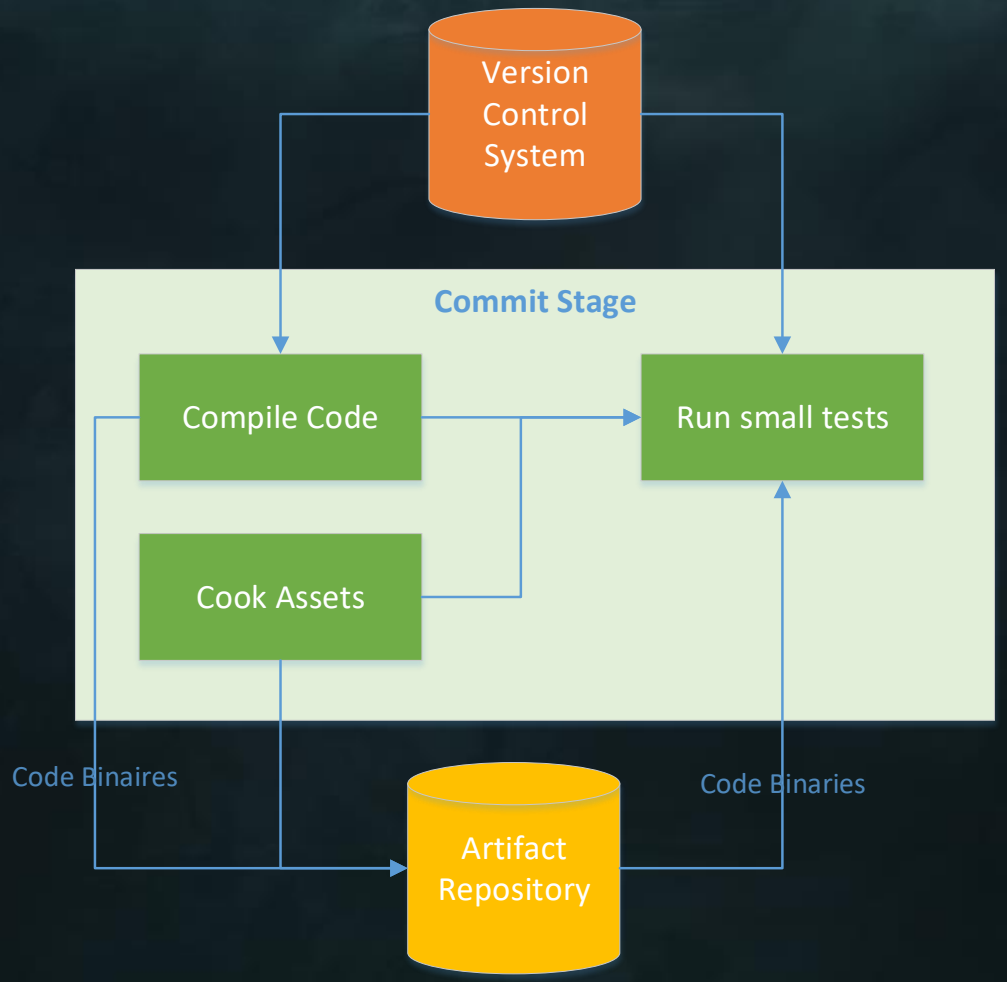
Challenges in adopting continuous Delivery

Challenges in adopting continuous Delivery

Adopting Testing mind-set:

- Started with a small core team who believed in this idea
- Gradually added more people to the team
- Having a separate prototype was crucial to setup the project correctly from the beginning
- Check Adequate tests added during code review

Commit Stage



Commit Stage

Developers wait for the verification before starting new work. Go to meeting, lunch or home

Having a Fast commit stage

Lessons :

- need to continuously improving otherwise it gets worst
- optimisation can lead to more complicated systems
- Monitor stability as you optimise
- Identify and Stop doing unnecessary work

Having a Fast commit stage

Compiling large C++ codebase takes a long time:

- Use a distributed build system
- Incremental build
- Build farm consists of 150 powerful physical PCs

Having a Fast commit stage

Identifying and prioritising most valuable tests

- Prioritise running tests that break more often but still run other tests at lower frequency
- Create a map between code and test, only run the tests that are affected by code Change

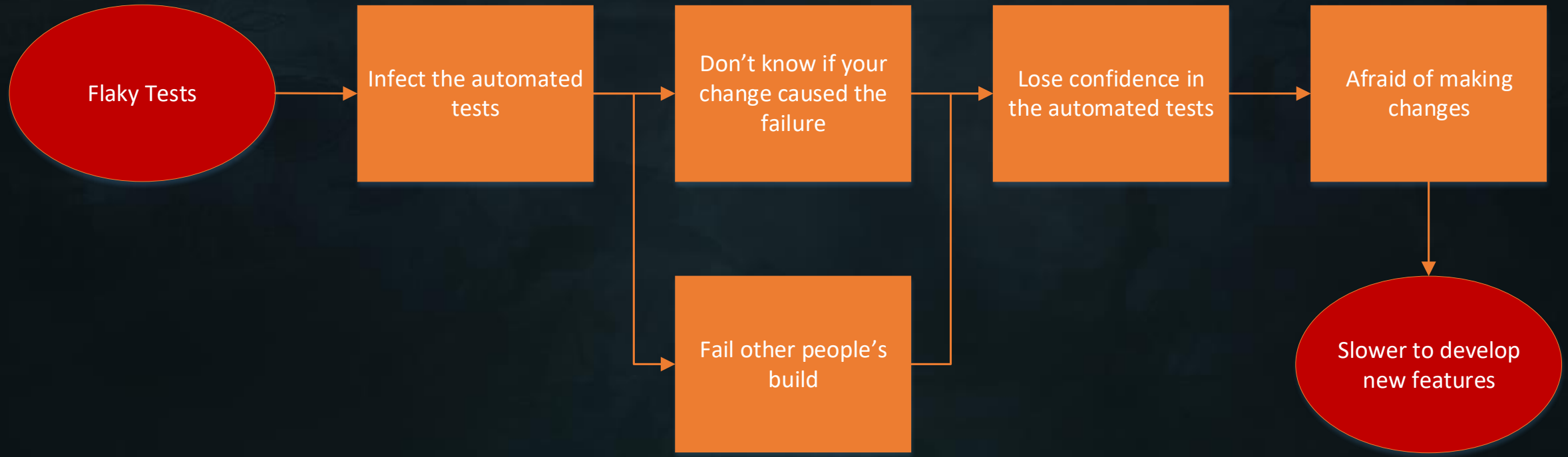
Having a Fast commit stage

Transferring multi GB files between agents in build farm:

- Fast network connection between agents
- Each Build agent has a local cache, don't have to download same build again
- Build Agents serve builds to each other (high network bandwidth)

Challenges in adopting continuous Delivery

Continuously identifying and eliminating flaky tests



Challenges in adopting continuous Delivery

Continuously identifying and eliminating flaky tests:

- Fewer deterministic tests much better than lots of tests that are flaky
- Don't let flaky tests infect the pipeline
- Quarantine flaky tests:
  - Fix them and move them out
  - Delete them

Common causes of flaky tests

- Using random waits, sleep for X seconds, common when testing async behaviour
- Tests not isolated, might pass or fail depends on tests that ran previously
- Relying on external dependencies such as a remote service

How much time to allocate for improvement

Three types of work:

1. develop feature
2. Unplanned and emergent work such as fixing bugs,
3. Reduce root cause of unplanned work (improvement work)

There's no rule on how much you should dedicate to each

Too much unplanned work means not enough time on improvement

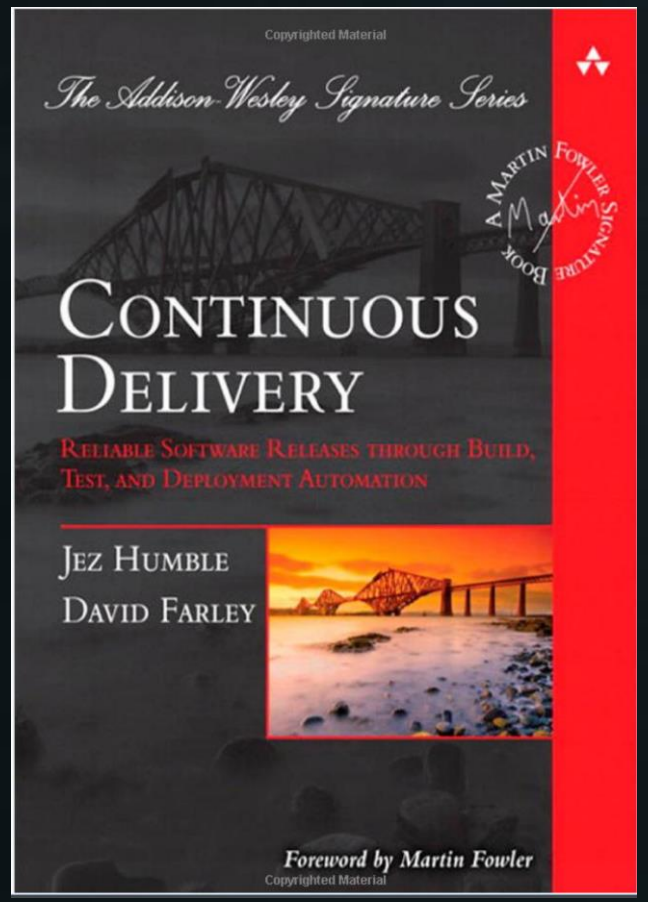
Challenges in Releasing weekly with confidence

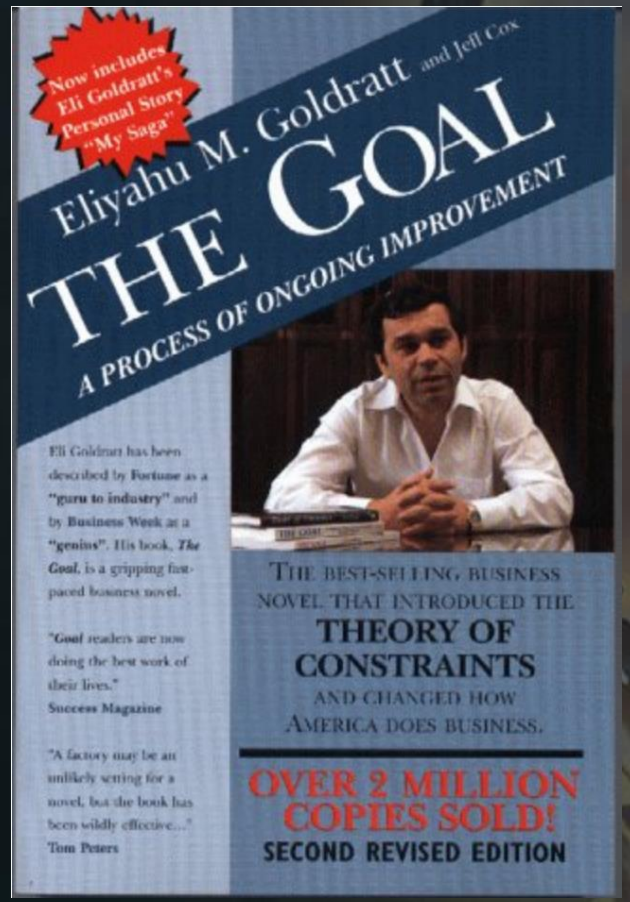
- Keeping patch size small
- Delivering new features regularly/weekly
- minimising the impact of something going
- Respond quickly to Incidents
- Certification process

Summary

- Work in small batches
- Release regularly and safely
- something will go wrong, fast response time
- Continuously improve
- we're not done with continuous delivery

References


Copyrighted Material
The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble
David Farley
Foreword by Martin Fowler
Copyrighted Material


Now includes Eli Goldratt's Personal Story "My Saga"
Eliyahu M. Goldratt and Jeff Cox
THE GOAL
A PROCESS OF ONGOING IMPROVEMENT
Eli Goldratt has been described by Fortune as a "guru to industry" and by Business Week as a "genius". His book, The Goal, is a gripping fast-paced business novel.
"Goal readers are now doing the best work of their lives."
Success Magazine
"A factory may be an unlikely setting for a novel, but the book has been wildly effective..."
Tom Peters
THE BEST-SELLING BUSINESS NOVEL THAT INTRODUCED THE
THEORY OF CONSTRAINTS
AND CHANGED HOW AMERICA DOES BUSINESS.
OVER 2 MILLION COPIES SOLD!
SECOND REVISED EDITION

Continuous Delivery Test

Thank you