Implementing DevOps using the Theory of Constraints

We want to deliver value to customers more frequently and reliably.

Constraint	Action - Exploit, Elevate, Remove
The time to implement code changes. The batch size is too large, and inhibits flow.	Exploit the Constraint: Make smaller changes, more frequently. Build the system after each change.
The time to build the changed code doesn't support the more frequent changes.	Exploit the Constraint: Automate the build process with a script or build tool.
It takes too long before one developer's change is accommodated by another developer. Sometimes developers are working with stale versions.	Exploit the Constraint: All developers pull from and integrate to the mainline of the code repository.
The time to integrate code changes can't support the more frequent integration. Sometimes one change conflicts with other functionality.	Exploit the Constraint: Use automated unit testing, specifically Test Driven Development, to reduce the time it takes to know if one change has violated the assumptions of other code. Run the unit tests after each build.
The time to deploy a changed system can't support the more frequent changes. Sometimes something goes wrong in the deployment process, and it takes even longer.	Exploit the Constraint: Script the deployment process so that it is repeatable and automated.
The time to verify proper business function and detect regressions can't support the more frequent integration. Sometimes some aspects of the desired functionality are missing or incorrect. Sometimes some functionality that did work correctly no longer does.	Exploit the Constraint: Use automated acceptance tests to know when desired business functionality has been achieved or has regressed. Run the acceptance tests after each successful unit test.
Automated tests don't detect unanticipated display and operation problems. Manual regression testing would have noticed such issues in the course of testing.	Elevate and Remove the Constraint: Perform Exploratory Testing on a less-frequent interval to notice these issues.
Sometimes the application behaves differently in one environment than another.	Exploit the Constraint: Do not rebuild the system for each environment. Build it once and store the deployable version in an artifact repository. Deploy the same artifact to all environments.
	Exploit the Constraint: Do not let the environments vary in an uncontrolled manner. Script the configuration of each environment. Have minimal differences between the environments. Going further, script the creation of each environment and recreate from scratch frequently (on a schedule, or with each deployment).
Frequent deployments to production cause noticeable outages during deployment. Any errors or mis-features that reach production cause turmoil for the users.	Exploit the Constraint: Use zero-downtime deployment and flexible release strategies (blue-green deployments, canary releases, dark releases)
Production issues, especially slow-downs and partial outages, may go unnoticed until users complain. This makes them slower to fix and greater in impact.	Elevate and Remove the Constraint: Add monitoring to enable viewing critical aspects. Record metrics to view how these aspects vary over time.

About the Theory of Constraints

The Theory of Constraints says that a system's throughput is, at any given time, controlled by its most limiting factor, or bottleneck. Improving performance at other points in the system is unlikely to improve system performance.

The Theory of Constraints offers a five-step process to system improvement:

1. **Identify** the current constraint that is limiting system operation.
2. **Exploit** the constraint--make improvements to the throughput at the constraint using existing
3. **Subordinate** other activities to support the constraint.
4. **Elevate** the costraint and take actions, perhaps restructuring the process, to eliminate it from
5. **Repeat** these five steps on the new constraint once this one is no longer the limiting factor.