CODE CLIMATE

Lean Experiments for Engineers

Improving Your Software Development Lifecycle (SDLC)

What Are Lean Experiments, within the context of SDLC?

Lean Experiments are a proven means of testing process improvements in a controlled, measurable way to verify their effectiveness before broader adoption. Inspired by Lean Manufacturing and Continuous Improvement, this practice is designed to deliver results faster, with less wasted time.. The Software Development Lifecycle refers most commonly to the process of developing and delivering software but for a growing number of organizations, refers more broadly to the process of delivering high value solutions to customers. Often, the goal is to deliver the most value, fast, at the lowest cost– but this is not easy for large organizations to achieve due to the high number of people performing work in often different ways as part of their own part of the process, and a number of variables ranging from changes in priority, staffing, the introduction of new technologies and more. This means changes to improve different parts of the process are rarely one size fits all, and can have negative consequences. Testing significant changes to improve the SDLC prior to scaling is a means of radically reducing risk.

When to Run SDLC Experiments

Not every change for improvement is a candidate for a Lean Experiment. Some changes just make sense to deploy – especially small changes that have been tested and proven by similar teams. Lean Experiments are most useful when you have an idea for a way to remove undesired friction in your SDLC, and/or improve a bottleneck, and you're uncertain of the impacts– good OR bad. So, you want to test it in a low-risk way before rolling it out at scale. Perhaps you're uncertain about the consequences of the change, or how quickly your change will yield desired results. Or maybe, you want to assess the potential uplift of what would be a significant process change before deploying it to understand if it's worth it. Lean Experimentation is a means of quickly isolating a solution worth deploying when there are risky & expensive assumptions.

While not every change requires a full experiment, the process of framing a change as a Lean Experiment is *always* useful to ensure there is a measured result from an effort. Setting up a Lean Experiment requires the establishment of a quantitative target for an important metric that indicates value is delivered, as well as a baseline, and a high degree of intentionality in selecting an approach. Consider: How many change initiatives or projects have you taken part in that haven't had these clear from the start– where you deployed a change, and carried on without knowing if it was a true success? By building a muscle of thinking of every change as a mini experiment, you'll avoid wasted money (and frustration!) that happens when there's no tangible evidence of a direct result of your work.

Where might you find good candidates for experimentation in the SDLC? Start by thinking about the end to end value stream, and where the bottlenecks exist:

- **PR Cycle Time:** Is the time from PR creation to merge too long?
- **Code Review Efficiency:** Are reviews taking too many iterations or too much time?
- **Bug Rate:** Do too many bugs slip through to production?
- **Development Velocity:** Are stories taking longer than expected to complete?
- **Merge Conflicts**: Do teams experience frequent merge conflicts?
- **Onboarding Time:** Are new developers taking too long to become productive?
- **Technical Debt Management:** Does addressing tech debt often compete to a problematic extent with feature work?

Every one of these challenges, and myriad others, has multiple potential approaches and solutions. Finding the right one starts with understanding the root cause of the problem, and the impact it has upstream and downstream in the SDLC.

Getting Started: What You'll Need

- Access to your team's metrics, which Code Climate provides
- A way to document your experiment (Jira, GitHub Issues, Notion)
- Buy-in from team members to try a new process
- Clear baseline measurements for current performance
- A defined timeframe for your experiment (typically 1-2 sprints)
- A plan for measuring results

Step-by-Step Guide to SDLC Process Experiments

Step 1: Clarify the Challenge

Ideally your org focus is to holistically improve Time to Value, inclusive of the entire end-to-end process starting with Idea, moving into Validated/Invalidated, then to Development, Delivery, Release and ending with User Adoption. However, your org may be focused primarily on a subset of this flow– for example, improving PR Throughput while maintaining or improving Quality. Ideally, you're clear before starting your experiments on what the higher level challenge is: by what percentage must we improve in a certain area to call the outcome a success? And how will it be measured? Without clarity on this you may have trouble prioritizing efforts and knowing when

you're done with improvement in a certain area. This higher level goal narrows the parameters you're working in.

Step 2: Identify Your Process Pain Points

Once you're clear on the challenge, identify specific process issues affecting your team's effectiveness:

1. Analyze your current metrics and identify bottlenecks, spotting values that are undesirably high or low. For example:
    - Is the median time PRs spend waiting for review very high?
    - Is the average number of review cycles high– more than 3?
    - Is there a relatively high proportion of bugs worked vs. new features developed?
    - Is the median time between 1st commit & PR Open, i.e. Time to Open, very high?

In Code Climate, you can inspect the data behind metrics down to the resource & activity level by drilling into aggregate metric values to see all the detail in Resource Explorer. This can be very useful to study patterns and understand issues, but it is not a substitute for a conversation.

2. Gather qualitative feedback from team members on why these bottlenecks may exist. Be transparent with your team members about the data and what you see, and use this to have objective & safe discussions about why these issues might exist. Ask more general questions, too, to get useful surrounding context:
    - "What part of our development process feels most painful?"
    - "Where do you feel you waste the most time?"
    - "What process issues cause the most frustration?"

In these discussions, you might hear some hypotheses based on individual experience:

- "Our PRs sit unreviewed for an average of 27 hours, and it may be because we only have two reviewers"
- "We typically go through 4+ review iterations before merging because the requirements are so unclear at the start."
- "We spend 40% of our sprint addressing bugs from previous sprints because of that "reduce the PR Cycle Time" effort we ran last year– without enough reviews."

Step 3: Convert Pain Points into Testable Hypotheses

In your discovery, you'll start to hear and form hypotheses for why the pain points exist, and what might solve them. What might not come naturally, but is the most important part, is how you'll know they're solved. This requires identifying a specific metric or two as indicator.

Generate a collection of hypotheses, using this format:

[Specific process change] will improve [specific metric] by [expected amount] to indicate we've solved [this problem]

Examples:

- "Implementing a 'PR buddy' system where each engineer is assigned daily review responsibilities will reduce average PR wait time from 27 hours to under 8 hours, which will indicate we've solved the wait time for PR Review"
- "Adding a PR template with a checklist will reduce review iterations from 4 to 2, to indicate we've solved the problem of too many review cycles which costs us PR Throughput and speed"
- "Adding pre-commit hooks for linting and formatting will reduce style-related review comments by 90%, to indicate we've solved the problem of style inconsistencies being an issue increasing review cycle time."
- "Implementing daily pair programming sessions will reduce our defect resolution rate by 30% to indicate we're on track to solve our overall quality issues."

Your hypothesis should be:

- Specific and measurable
- Limited in scope
- Time-bound
- Focused on one process change at a time

In the examples above, you might notice how one experiment can cause effects elsewhere in the SDLC that aren't always optimal. For example, taking focused action on reducing bugs by introducing pair programming is likely to increase overall development time initially. So it's ideal to identify 2-3 "secondary metrics" that you want to keep an eye on to better understand potential undesirable impacts as part of these quick, small tests. **This is, in fact, one of the biggest benefits of Lean Experimentation.** Save time & money by finding out what might go wrong, quickly, then test modifications before scaling to ensure the best outcome possible.

Step 4: Design Your Process Experiment

Once you've selected the best hypothesis worth testing first, design an experiment that will test your hypothesis with minimal disruption. What's key is to isolate and control the variables. You

want the only thing that could impact the metric in your hypothesis to be the solution you're testing. There are often many variables to consider, for example: complexity of work week to week; changes in the backlog; staffing changes. You'll want to tightly control the conditions of your test to get the best, most reliable result. Some best practices here:

- **Set a timebox**. You should absolutely set a timebox for your experiment, keeping it as short as possible– no more than 2 iterations or sprints is ideal. Any longer, and you're increasing the cost of delay of scaling a solution that works, which you'll only find through quick/light tests.
- **Schedule your experiment to avoid variables**. You want to run an experiment with a team, or teams, that experience no other changes that would impact their SDLC other than your solution. This is aspirational and hard to 100% achieve, but the more you reduce the variables, the more reliable your result which can be especially useful if, for example, your first solution is a clear winner according to your metric & target and it's time to scale. No need to waste time testing others! Consider: Are you running into a code freeze period? Is the team/teams you're planning to involve about to encounter some increasingly complex work? Are they working through some dependency challenges? Try to identify and manage as many variables as possible.
- **Identify: one team or two?** Perhaps you want to test a solution on one team, looking at results before and after. Or, perhaps you want to include two teams, one serving at the control, and the other testing your solution so you can see the contrast between teams given your target metric.
- **Have a rollback plan.** If you happen to learn that the change you're testing is disruptive and/or results in some negative impact that is serious and obvious, you'll want to be able to efficiently end the experiment. Have a lightweight plan to do this in advance.

Step 5: Set a Target, Establish Your Baseline

You may do this before or after designing your experiment, but it's a critical step: establishing a target improvement in your metric, and establishing your baseline at the start of your experiment.

- Establish a target value. By how much do you aim to change your metric? What do you think is possible? You may have isolated 3 important problems that are causing your target metric to be sub-optimal, with potentially 3 different solutions. In this case it'd be unrealistic to expect that you'll see a huge improvement based on one solution. Avoid getting paralyzed by trying to set the "perfect" target value; you don't really know the impact, after all, which is why you're performing the test. However, you should know which metric will change, and you should know the direction it should change in to confirm your solution is a good one. So think of the target as a rough goal. Remember that the primary goal of the experiment is to learn fast.

- Establish a baseline. Be sure to record the starting metric value at the start of your experiment, so you can assess whether it changes following the experiment. This may seem so obvious, but many teams miss this part!

Step 6: Document & Communicate Your Process Experiment Plan

You've likely already done a lot of the work here: you've formed a hypothesis, designed a test, set a target and sorted the baseline. Be sure to give the people involved in the experiment advance notice of the details of the experiment, too. This is another thing that may seem obvious, but often gets missed. When the people involved in an experiment and/or the people they work with do not know and don't have advance notice, they may make other plans. Their priorities might shift, they may decide to take on some complex tech debt or other complicated work, or they may even take it upon themselves to solve the same problem you are. The solution is fairly simple: communicate in advance. Give people involved and with a stake in the experiment at least a week's worth of notice.

Here is an example of an Experiment Plan that can easily be distributed and discussed:

Process Experiment: PR Buddy System

Problems

- PRs wait an average of 27 hours for first review
- Engineers report context switching as major problem when reviews interrupt focused work

Hypothesis

- We believe that by implementing a rotating "PR buddy" system where each engineer is assigned daily review responsibilities will reduce average PR wait time from 27 hours to under 8 hours without negatively impacting engineer productivity.

Metrics

- Primary: Time to First Review (we want this to go down)
- Secondary: PR cycle time (we want it, at worst, to hold stead though we expect it to increase slightly)

Experiment Details

- Duration: 2 weeks (May 5-19)

- Each day, one engineer will be designated "PR buddy" whose primary responsibility is timely review of all new PRs
- Engineers will take turns being PR buddy according to this schedule: [schedule]
- PR buddy expectations: Review PRs within 4 hours of submission
- During PR buddy day, the engineer is not expected to make significant progress on feature work

Success Criteria

- Success: Average time to first review under 8 hours
- Partial success: Average time to first review 8-13 hours
- Failure: Average time to first review over 13 hours or significant negative impact on PR Cycle Time. If this happens, run post mortem and assess why

Rollback Criteria

We will end the experiment early if:

- Multiple engineers report significant disruption to their work
- Critical deadlines are at risk due to reduced development capacity

Step 7: Execute Your Process Experiment

Time to run your Experiment! Be sure to monitor results as it runs, not only to see how it's progressing but also to determine whether a rollback is needed. Capture observations through the experiment, both noted in the data and discovered through interactions with the people involved in your change.

Execution tips:

- Send calendar invites for the experiment start/end dates
- Create visible reminders of the new process
- Designate a point person to answer questions about the experiment
- Create a feedback channel for team members to report issues
- Hold weekly check-ins to ensure the experiment stays on track, share quantitative results and gather feedback

Step 8: Analyze Results, and Make Decisions

After the experiment period:

1. Compile all metrics and compare to baseline
2. Gather qualitative feedback from all team members
3. Evaluate results against your success criteria:
    - **Successful:** The process change achieved the desired improvement
    - **Partially successful:** The process showed some improvement but didn't meet full success criteria
    - **Unsuccessful:** The process didn't improve the target metrics or had negative side effects
4. Document your findings:
    - What worked well?
    - What unexpected challenges arose?
    - What did the team like or dislike about the new process?
    - Were there any unintended consequences (positive or negative)?
5. Decide on next steps:
    - **If successful:** Formalize the process change as team standard and consider scaling or, if needed, run 1-2 more tests to eliminate any additional risky unknowns
    - **If partially successful:** Refine the process and run another experiment
    - **If unsuccessful:** Try a different approach to address the same pain point
6. Create an action plan for implementation or follow-up experiments

Example Experiment: Reducing PR Review Time

Here is an example of an experiment to address a common bottleneck: PRs stuck waiting for review for too long. This can be for a variety of reasons, but in this example, the team has isolated the biggest reason as there simply not being enough accountable reviewers.

Pain Point: PRs wait an average of 27 hours for first review, causing context switching, merge conflicts, and lower throughput.

Hypothesis (with target & baseline): Implementing a rotating "PR buddy" system where each engineer is assigned daily review responsibilities will reduce average Time to First Review from 27 hours to under 8 hours.

Experiment Plan:

- Duration: 1 sprint (2 weeks)
- Each day, one engineer serves as "PR buddy" with primary focus on timely code reviews
- PR buddy rotates daily according to predetermined schedule
- PR buddy aims to review all new PRs within 4 hours of submission
- Reduced feature development expectations for PR buddy on their assigned day

Metrics:

- **Primary:** Average Time to First Review
- **Secondary:** Average PR Cycle Time

Results:

- Average Time to First Review dropped from 27 hours to 5.3 hours
- Avg PR Cycle Time reduced from 60 hours to 40
- Developer satisfaction also increased due to faster feedback and fewer merge conflicts

Decision: Deploy to 5 additional teams and assess Defect Resolution Rate as well as overall PR Cycle Time to assess whether there are important downstream impacts that would come with deploying this change at scale.

Example Timeline

Here is an example of how you might break up the steps for your next experiment over 6 weeks:

Week 1: Preparation

- Monday: Identify process pain point and current metrics
- Tuesday: Formulate hypothesis and success criteria
- Wednesday: Design experiment details and measurement approach
- Thursday: Document experiment plan and share with team
- Friday: Set up measurement tools and baseline metrics

Week 2-5: Execution

- Day 1: Kick off experiment in team meeting
- Throughout: Collect daily metrics and observations
- Weekly: Hold check-in to address any issues

- End of Week 3: Conclude experiment phase

Week 6: Analysis and Decision

- Monday: Compile all metrics and feedback
- Tuesday: Analyze results against success criteria
- Wednesday: Prepare findings and recommendations
- Thursday: Present results to team and decide on next steps
- Friday: Document decision and plan

Additional Reading

- Running Lean by Ash Maurya- A practical guide on lean experimentation
- The Lean Startup by Eric Ries - Foundational book on product practices inspired by Lean Manufacturing
- Toyota Kata by Mike Rother - A great overview of continuous improvement and how it was applied at Toyota
- Lean Enterprise Institute - A nonprofit focused on helping organizations apply lean principles to work to reduce waste & increase value delivered
- Team Topologies - Organizing team structures for effective delivery

Code Climate Powers Up Your Experiment Practice

Code Climate provides insights that power experiments at Fortune 100 enterprise organizations. With Code Climate, you can take action to improve in areas that matter most to you, with confidence.

Code Climate was founded in 2011 by engineering leaders who were frustrated by the lack of visibility into data that would enable them to make informed improvements. Today, engineering executives from Global enterprise organizations in industries such as Hospitality, Healthcare, Retail, and Technology partner with Code Climate to tackle critical business challenges. Through our personalized approach, expert guidance, and a custom-designed insights platform, Code Climate delivers tailored, actionable insights that empower engineering executives to make data-driven decisions.

We believe value only comes when engineering leaders act on important insights. We also believe value sooner is better– and quick, light experiments are the best means to effect measured improvement week over week, month over month. Given our background in Lean Startup, we're experienced at helping organizations become more productive, higher performing, and thriving.

If you'd like to learn more, contact us at support@codeclimate.com. We look forward to hearing from you!