The following narrative is my response to the question highlighted in blue below:

What is the most inventive or innovative thing you've done? It doesn't have to be something that's patented. It could be a process change, product idea, a new metric or customer facing interface – something that was your idea. It cannot be anything your current or previous employer would deem confidential information. Please provide us with context to understand the invention/innovation. What problem were you seeking to solve? Why was it important? What was the result? Why or how did it make a difference and change things?

As an engineering leader at Cisco, my most innovative accomplishment was the operationalization of the componentization program for the platform independent (PI) software team. For years, the PI software team had struggled to deliver software components in four main software release trains. The release trains had drifted apart over time, quality had declined, and maintenance costs were not sustainable. Customers who owned products that ran on different code releases were exposed to feature gaps, missing bug fixes, and operational inconsistencies. The goal of the componentization program was to roll out a single PI component source code which would be used in each of the four release trains to lower engineering costs and increase customer quality.

The componentization program was a huge undertaking for the PI organization. Over 100 engineers were assigned to the first phase which was to deliver an initial group of 10 'hot' components. Early in the program, I developed the PI component integration strategy which provided an opportunity for each component team to demonstrate incremental results during the long development cycle. One software release train, led by the MCP product team, was selected for the initial component integration. Based on my prior relationship with the MCP team, I worked with them to establish a process to integrate incremental versions of component code into their release train. They did not trust the PI team's ability to perform these operations. We needed to build trust if we were ever going to scale up.

Despite my strong interest in the program, my initial involvement was limited. My team was not responsible for a 'hot' component, so my manager wanted to limit my time in the program. I largely disengaged from the program for about 15 months while another manager headed the daily program execution. That manager left the company just as the program reached the final stages of integration. My manager also moved on to a new role, and a new manager took his place. My new manager assumed sponsorship for the program. He asked me to take over its overall execution. His guidance to me was simply to "make it work."

The MCP team had a high level of distrust in the PI organization. They came to Cisco as an acquisition and had their own separate compensation package that was tied to financial targets. They preferred to take their own copy of the PI code and modify it to suit their needs. It was difficult to convince their technical leaders that the merits of componentization were in their best interest. I knew the MCP team required assurance that the PI team was doing everything it said it would do at each stage of the release process. The PI team needed to establish self-policing practices to ensure that the MCP team's expectations were being met, otherwise the PI team would not achieve operational autonomy.

Since component releases were effectively small software releases, many of the existing software release processes needed to be translated to the component level. We needed quality gates, release branch plans, exception processes, and a component creation process. These processes would be created as needed due to time pressure.

The team was nearing the first release when I re-engaged in the program. The first thing I created was a per-component throttle pull gate review. I determined that we needed to review the following: test execution results; defect backlog; disposition of static analysis warnings; and defect gaps relative to prior releases. This was the basis for the initial set of reviews. I reasoned that the process could be changed if needed for future releases. To aid the teams in completing this process step, a template document was created along with step by step instructions. Automated tooling existed for collecting the data. The teams simply had to gather it all in one place for quick review. For the first release, the MCP team participated in these reviews.

To counter the MCP team's mistrust in our ability to manage a large number of component releases, we needed a means to document those component release plans for the subsequent time-based release's execution commit gate review. One of the larger component teams had created an easy to understand document for their own use. I decided to leverage this as a template but to permit each team to provide the same planning information in alternate formats if it better suited their scale. I created a roll up spreadsheet that each team linked to their release branch plan document, as well as provided additional high level planning information. The roll up spreadsheet was then linked to the overall gate review document.

As the initial phase of the program neared completion, more component teams were eager to jump on the componentization band wagon. With high levels of mistrust from the product teams, care needed to be exercised each time the program expanded.

I created a process gate review for the creation of new components to assure that there was sufficient return on investment from the effort.

Our release engineering team had relationships with several of the platform teams. I established a weekly meeting with the lead representatives from each of the platform facing teams. I worked with this team to establish a late feature exception process similar to the one used for standard releases. Giving them a voice improved their willingness to engage in the component process definitions. It also helped pave the way for integration into the other platform trains.

The initial release shipped on time and with few quality or process issues attributed to the componentization effort. The product team trusted our throttle pull gate review entirely. Component teams were subjected to lower degrees of scrutiny in subsequent releases.

Our second componentized release train shipped nine months later. Planning and execution with this product team was more straightforward due to our proven results. No process changes were needed to accommodate the requirements of this team, however document templates and tooling automation were improved.

It took two years to release components on the remaining two product trains, but this was in large part due to differences in their release process. Our component release plans, throttle pull, and late feature exception reviews were critical items for acceptance from these teams. These content-based releases lagged significantly behind the time-based releases, resulting in a need for much longer planning horizons.

Within a year we had built enough trust with the product team that some component teams moved to self-management. Two years later, all components were self-managed and the central component team was disbanded, greatly reducing overhead. The component count grew from the initial 10 to over 60 within four years. In hindsight, I could have pressed for disbanding central management of the program sooner. The experienced teams had what they needed to self-manage, and sufficient documentation and tribal knowledge was in place to help bring others up.

During the first year of program operation, the sustaining investment for the hot components dropped from nearly 60% to less than 30%. Defect backlogs, which had been running as high as 5 times the incoming weekly bug rate for some teams, were held at an organizationally mandated 3 to 3.5 times the incoming weekly bug rate. The investment was lower, and the quality was substantially higher.