SonarQube

1)개념 : 정적 코드 분석 도구 중 하나
: 실제 실행 없이 컴퓨터 소프트웨어를 분석하는 것

대부분의 경우에 분석은 소스코드의 버전 중 하나의 형태로 수행
정적 분석은 주로 개발 단계에서 소스 코드의 구조적인 문제나 실수를 찾아내는 데 사용 (동적 분석은 테스트나 모니터링할 때 사용)

2) SonarQube 를 선택한 이유
-> 레퍼런스가 많고 Github, Jenkins 와의 연동을 통해
자동 정적 코드 분석을 구성할 수 있기 때문

3) Sonarqube 에서 관리해주는 소프트웨어 품질

- Code Smell : 심각한 이슈는 아니지만 베스트 프렉티스에서 사소한 이슈들로 모듈성(*modularity*), 이해 가능성(*understandability*), 변경 가능성(*changeability*), 테스트 용의성(*testability*), 재사용성(*reusability*) 등이 포함된다.
- Bugs : 일반적으로 잠재적인 버그 혹은 실행시간에 예상되는 동작을 하지 않는 코드를 나타낸다
- Vulnerabilities : 해커들에게 잠재적인 약점이 될 수 있는 보안상의 이슈를 말한다. SQL 인젝션, 크로스 사이트 스크립팅과 같은 보안 취약성을 찾아낸다.
- Complexity : 코드의 순환 복잡도, 인지 복잡도를 측정한다.
- Unit Test : 단위테스트 커버리지를 통해 단위 테스트의 수행 정도와 수행한 테스트의 성공/실패 정보를 제공한다.
- Duplications : 코드 중복은 코드의 품질을 저해시키는 가장 큰 요인 중 하나이다.
- Size : 소스코드 사이즈와 관련된 다양한 지표를 제공한다.

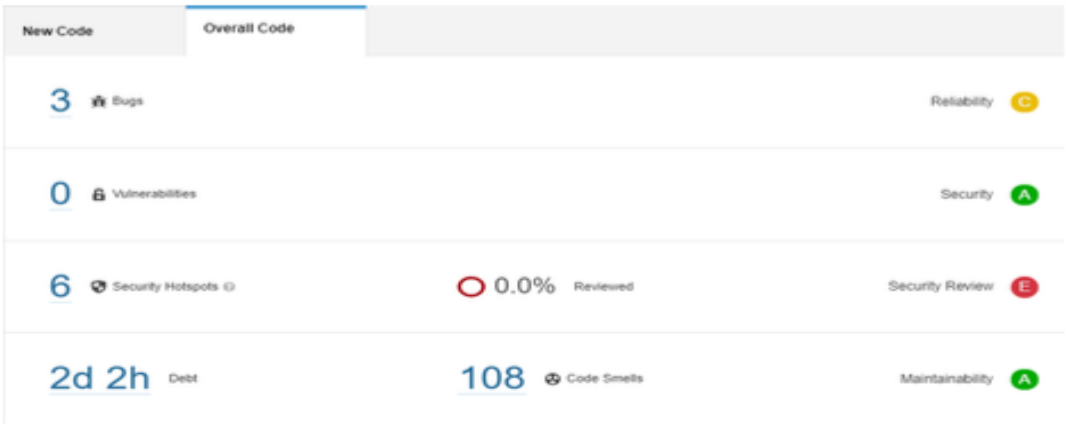


그림 1) SonarQube 로 코드 분석을 시행한 이후의 화면

-> 미리 정의되어 있는 빌드 규칙에 따라 프로그램이 빌드 될 때마다 자동으로 코드 분석을 수행해준다. SonarQube 는 프로그램에 존재하는 문제점들을 명료하게 짚어주고, 각 문제점들에 대한 몇 가지 해결책을 제안해주기 때문에 쉽게 프로그램을 개선할 수 있다.

그림 2) SonarQube 의 Code Smells 분석

-> 각 항목을 클릭하면, 해당 항목에서 어떤 문제가 있는지 나열되고, 각 문제들이 어째서 문제인지, 어떻게 해결해야 하는지에 대해 상세히 확인할 수 있다

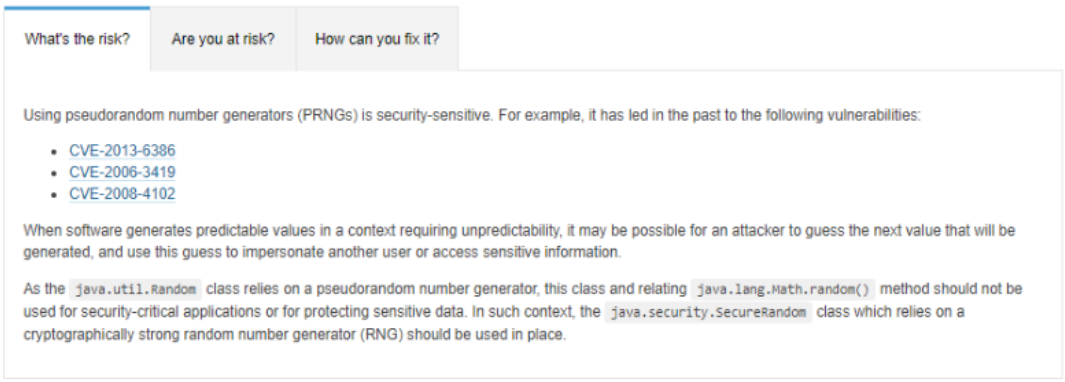


그림 3) Security Hotspot 화면

-> 과거에 해당 문제가 일으킨 사례 또한 확인할 수 있다.

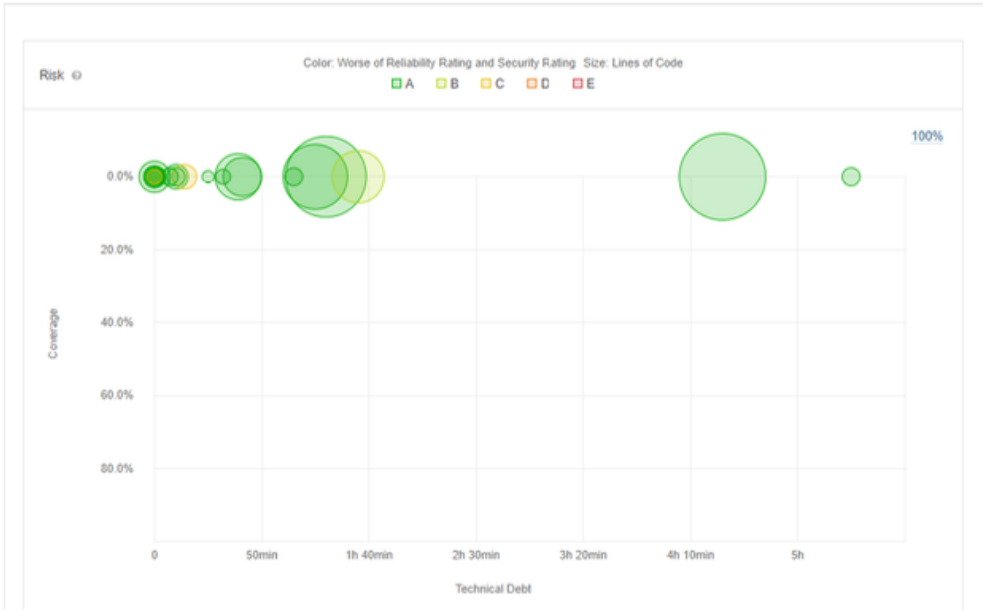


그림 4) Project Measure 탭

->

SonarQube 는 그래프나 색채 등 시각적 자료를 통해 프로그램의 품질에 대해 직관적으로 이해하는 것을 돕는다.

원의 크기는 코드 라인 수를 의미하며, 색상은 코드 안전성 및 안정성을 의미한다. y 축은 테스트가 얼마나 프로그램을 커버하고 있는지 나타내는 code coverage 이며 그래프 상에서 아래에 원이 위치할수록 좋다. x 축은 기술 부채로, 해당 파일을 리팩토링 하거나 다른 곳에 적용하기 위해 필요한 비용을 시간으로 나타낸 것이다.

SonarQube를 사용할 때, 예상되는 장점

소프트웨어 개발은 크게 기능, 비기능 요건을 충족하도록 구현해야 하는데, 이 두가지 모두를 충족하지 못하면 문제가 발생한다.
기능은 충족했지만, 성능, 보안, 확장성 등의 수많은 비기능 요건을 충족하지 못해서 재작업이라는 추가 비용이 발생되는 경우가 많이 있다.

비기능 요건이 기능 요건에 비해 다루기 어려운 이유는
기능 요건에 대해서 고객과 충분히 협의나 협상이 가능해서 범위를 조정할 수도 있지만, 비기능 요건에 대해서는 기본적인 베이스라인이 충족해야만 해당 기능 요건을 실행할 수 있는 토대를 마련할 수 있기 때문에, 고객과의 협의 폭이 크지 않다.

기능 요건은 개발을 통해 고객에게 충분히 가시적으로 보여주는 영역이 있는 반면, 비기능 요건에 대해서는 가시적으로 시각할 수 있는 범위가 제한적이고, 별도의 비용이 발생된다.

비기능 요건인 소프트웨어 품질은 많은 부분에 있어 개발에 대한 부담으로 작용하는게 사실이고, 소스 코드의 분량이 늘어날수록 품질은 그만큼 더 안좋아지게 되며, 이러한 상황을 막기위해 품질 활동이 뒷받침되어야 변경이 전의 상태 수준으로 맞출 수 있다.
분석결과를 토대로 코드 스타일도 점검하고 가독성 높은 코드를 짤 수 있고, 잠재된 버그와 성능적인 이슈를 미리 확인할 수 있어 효율적인 코딩을 할 수 있다.

SonarQube는 이와 같은 상황에서 적은 시간과 비용으로 비기능 요건에 대한 가시화를 충족시킬 수 있는 도구로 작용할 수 있다. 품질 관련 활동을 SonarQube가 지속적으로 수행해주어 소프트웨어 품질이 유지 및 개선될 것이다.

Metrics

1)Spring Burn Down Chart

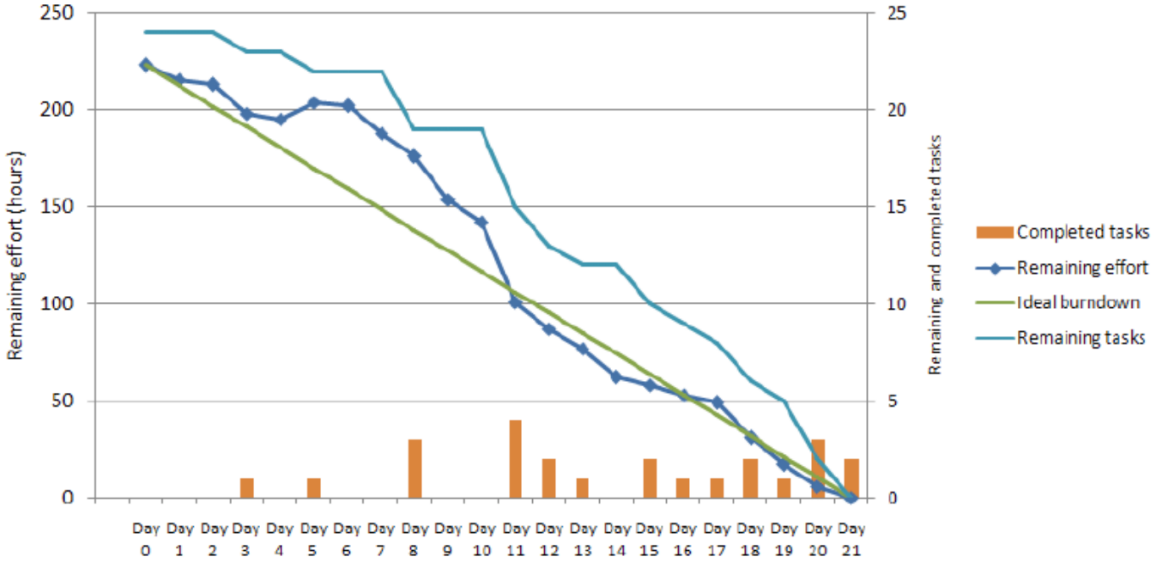
특정 작업 또는 “스프린트”에 남은 작업 측정한다.
Scrum agile 개발에서 중요한 역할을 담당하지만, 프로젝트 관리 스타일에 관계없이 유용하다. 팀 구성원들이 번다운 차트를 이용해 프로세스를 쉽게 확인 및 이해하고 그에 따라 조정해 목표를 달성할 수 있기 때문이다.

애자일 소프트웨어의 12 가지 원칙 중 하나는 '최고의 아키텍처, 요건, 설계는 자체적으로 조직되는 팀에게서 나온다'는 것이다. 그러나 스크럼과 칸반 등 애자일을 도입한 대부분 기업은 몇 가지 중요한 프로세스 엄격함과 의식을 강요한다. 예를 들어, 많은 기업이 애플리케이션 릴리즈의 비즈니스 영향, 품질 및 신뢰성을 개선하기 위해 스토리 포인트 추정, 아키텍처 표준 및 릴리즈 관리 분야를 포함한 애자일 기획을 한다.

대부분의 팀은 지라 소프트웨어(Jira Software) 또는 애저 데브옵스 같은 애자일 툴을 사용해 애자일 팀 간의 백로그, 스프린트 및 협업을 관리한다. 이들 툴의 주요 목적은 애자일 팀원 및 여러 개의 애자일 팀 간에 요구사항, 스프린트 상태, 워크플로우 및 협업을 중앙에서 관리하는 것이다. 따라서 조직이 이러한 툴을 더 엄격하게 사용할수록 리더와 팀이 문제를 식별하고, 상태에 대해 이해관계자에게 보고하고, 실행을 개선하는 데 더 많은 도움이 된다.

이런 툴을 통해 만들 수 있는 가장 유용한 보고서 중 하나가 번다운 보고서다. 애자일 관행으로 인해 제품 소유자는 고객 피드백에 따라 백로그의 우선순위를 다시 정할 수 있다. 갠트 차트(Gantt charts) 같은 전통적인 보고서는 애자일 실행의 이런 유동성을 포착하지 못한다.

번다운 차트의 기본은 완료된 작업, 범위에 추가된 새로운 작업 및 기타 범위 변경을 설명하는 것이다. 번다운 차트는 팀이 목표를 향해 나아가는 방법을 간단한 그림으로 보여준다.



1. 타임라인 설정

- **스프린트 길이 결정:** 지속가능한 속도를 유지하기 위해 스프린트 길이는 며칠에서 몇 주로 다양하게 설정할 수 있습니다. 프로젝트 매니저는 전체 팀의 정보를 취합한 후에 타임라인을 결정해야 합니다.
- **평가 횟수:** 검토 간격은 귀중한 정보를 제공할 정도로 충분하면서도, 필요한 경우 대응 또는 의미있는 조정을 할 수 있을 정도로 짧게 정하세요. 단기 스프린트의 경우 신속한 일일 체크인이 필요할 수 있습니다. 장기 스프린트의 경우 매주 또는 격주 체크인이면 충분할 수 있습니다.
- **어려움 예측:** 일부 프로젝트의 경우 목표를 달성하는 데 더 오랜 시간이 걸려 번다운 차트를 쓸모없게 만들 수 있습니다. 예를 들어, 복잡한 웹 및 IT 관련 프로젝트에서 팀 구성원들이 보다 복잡한 작업에 도달하는 동안 시간이 멈춘 것처럼 보일 수 있습니다. 이는 일일 또는 주간 회의에서 계획을 세우기 좋은 도전과제입니다.

2. 작업 수행

- **자원 할당:** 미리 결정한 부분만큼의 작업을 수행할 팀과 개별 참가자를 할당하세요.
- **명확한 작업:** 작업은 목표를 중심으로 명확하고 구체적이며, 측정 가능해야 합니다. 예를 들어, “콘텐츠 작성”이라는 제목의 과제를 생성하는 대신에 다음과 같이 세부적으로 나누어 작성할 수 있습니다. “웹 사이트 라이프스타일 섹션 방문자 수를 늘리기 위해 홈 데코 콘텐츠 15 페이지 작성”. 이러한 세분성을 통해 모든 사람이 함께 목표를 위해 노력하도록 합니다. 이는 또한 작업에 대한 소유 의식을 갖도록 해줍니다.
- **목표 지향:** 팀 구성원들은 달성한 목표의 명확한 비전을 가지고 작업 완료에 따른 영향을 이해해야 합니다. 완료일에 스프린트 검토를 수행함으로써 이를 실현할 수 있습니다. 여기서 창조적인 단계와 발생하는 문제에 대해 논의할 수 있습니다. 검토에는 작업 소유주, 프로젝트 매니저 및 기타 관련된 팀 구성원이 포함되어야 합니다. 스프린트가 조기에 완료되는 경우 차트에 백로드된 항목을 추가하는 기회에 대해 논의할 수 있습니다.

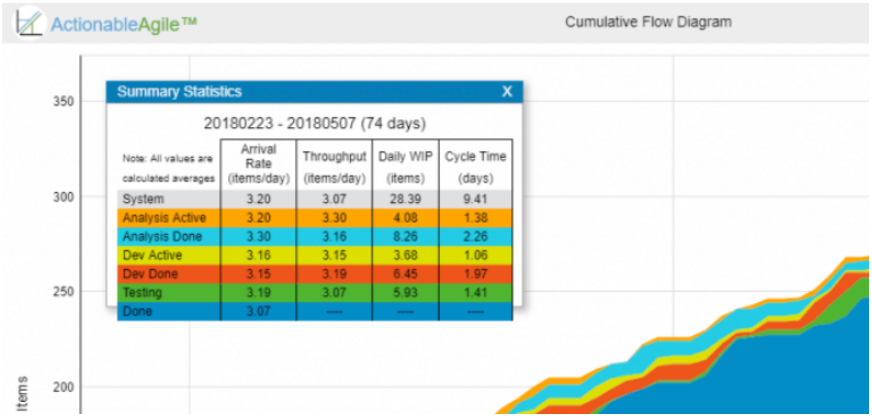
3. 정기적인 검토

- **체크인:** 조기에 자주 동기화하는 것은 팀을 다시 그룹화하고, 진행률을 표시하고, 즉시 피드백을 제공하는 데 도움이 되는 경우가 많습니다. 신속한 모임을 통해 원동력을 계속 유지할 수 있고, 몇 주를 기다리지 않고도 병목 현상과 격차를 해결할 수 있습니다.
- **일일 또는 정기적인 스탠드업:** 모든 사람에게 최신의 핵심 정보를 제공하는 신속하고 간단명료한 회의입니다. 오랫동안 토론을 지속하기 위한 시간이 아닙니다. 더 큰 문제가 발생하는 경우 적절한 이해당사자들과의 회의 후 논의를 지속할 수 있습니다. 짧고 간단명료하게 유지하는 것이 모든 사람이 동일한 방식으로 업데이트를 제공하도록 하는 데 도움이 됩니다.

2)Work in Progress (WIP)

스크럼 팀이 작업 흐름을 형성하는 데 도움이되는 "제약 조건"으로 사용하는 정책
칸반은 Work In Process 를 제한하여 애자일 방법론을 실행한다.
WIP 는 현재 수행하는 작업(work) 항목을 시각화하여 나타낸다.
WIP 제한의 목표: 실제 WIP (WIP)의 양을 줄이는 것으로, 진행중인 작업을 제한해서 모든 작업이 빠르게 완료하면 팀의 Throughput 증가한다. (처리량은 Velocity 와 다소 유사)

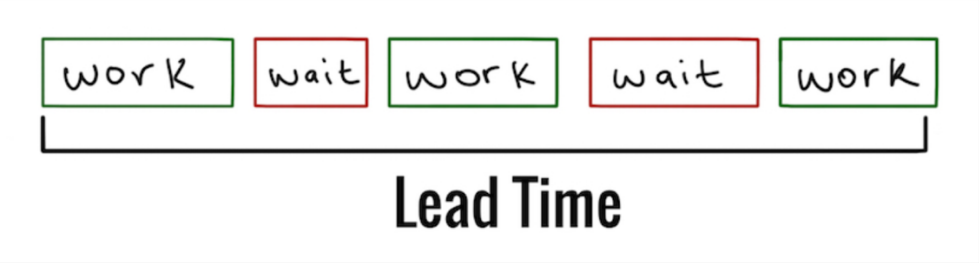
Kanban Board (WIP 시각화 도구)
1) Workflow 가시화
- 일을 작게 분할, 카드에 기록하여 보드에 게시
- 단계를 알 수 있도록 Flow 별 단계 기록
2) WIP 제한
- Workflow 상에서 동시에 진행될 수 있는 항목을 제한한다
3) 리드 타임 측정
- 한 항목을 완료하는데 걸리는 평균시간, 타임을 산정한다
- 예측 가능하고 소요시간을 최소화하기 위해서 프로세스를 최적화 한다



https://kissflow.com/project/agile/wip-limits-in-kanban/

3) Flow Efficiency

리드 타임을 구성하는 두 가지 기본 구성 요소인 작업 시간과 대기 시간을 검토한다.

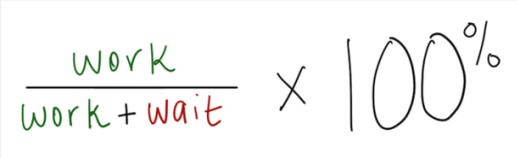


한 번에 한 가지 작업을 수행하고 중단되지 않는 한 리드 타임에는 이러한 구성 요소가 모두 포함된다. 대기 시간은 종속성, 우선 순위 변경, 너무 많은 작업 진행 등 여러 가지 이유로 발생할 수 있다.

목표 : 작업을 얼마나 효율적으로 처리할 수 있는지 모니터링 할 수 있다.

흐름 효율성 계산

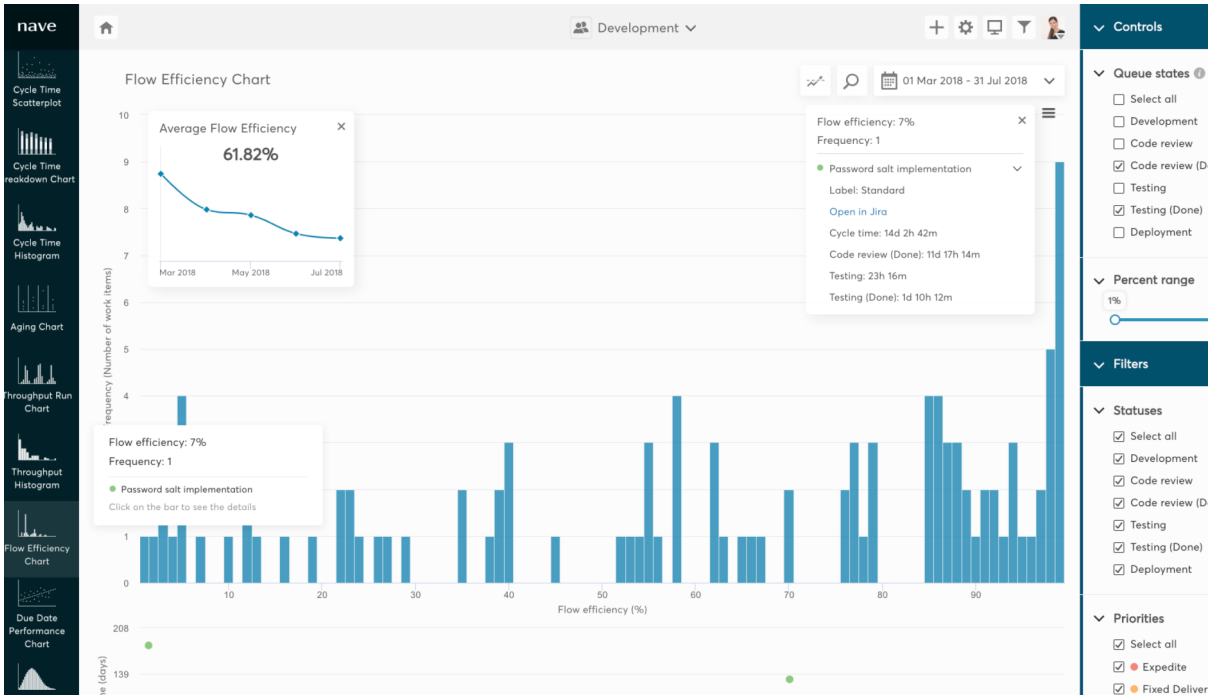
흐름 효율성을 계산하는 핵심은 작업이 대기하는 데 걸리는 시간을 아는 것이다.



단일 요청에 대해 흐름 효율성 측정을 수행 할 수 있지만 특정 기간에 완료된 모든 요청에 대해 프로세스의 흐름 효율성을 측정하려는 경우가 훨씬 더 많다. 따라서 해당 기간에 완료된 항목에 대해 다음 정보가 필요한다.

1. 전체 리드 타임 (작업 + 대기 시간
2. 활성 작업 시간 (대기 시간은 포함하지 않음)

그런 다음 활성 작업 시간을 전체 리드 타임으로 나누어 흐름 효율성을 계산한다. 해당 방정식의 결과에 100 %를 곱하면 결과는 주어진 시간 창에 대한 흐름 효율성이다.



-> Jira Flow Efficiency Chart