실패를 자산으로 바꾸는 법

포스트모템이 만드는 엔지니어링 문화

ThakiCloud (Hyojung Han)

실패를 자산으로 바꾸는 법

포스트모템이 만드는 엔지니어링 문화

ThakiCloud (Hyojung Han)

Contents

1 장애를 숨기는 조직은 같은 장애를 두 번 겪는다

새벽 세 시에 알람이 울린다. 결제 API가 30분째 5xx를 뱉고 있다. 당직 엔지니어가 급히 롤백하고 서비스는 복구된다. 다음 날 아침, 팀장이 묻는다. "어제 무슨 일이었어요?" 여기서 조직의 성격이 갈린다. 어떤 팀은 "제가 배포 전에 스테이징 테스트를 건너뛰었습니다, 죄송합니다"로 대화가 끝난다. 어떤 팀은 "왜 스테이징 테스트를 건너뛰어도 배포가 막히지 않았을까요"로 넘어간다. 앞의 대화는 사람을 처벌하고 끝나고, 뒤의 대화는 시스템을 고친다. 이 책은 뒤의 대화를 만드는 법을 다룬다.

1.1 비난이 정보를 막는 방식

장애 대응에서 가장 값진 자원은 정보다. 무엇이 실제로 일어났는지, 어떤 신호를 놓쳤는지, 왜 그 결정을 그 순간에 내렸는지. 이 정보는 대부분 당사자의 머릿속에만 있다. 그리고 그 정보를 꺼내는 유일한 방법은 당사자가 자발적으로 말하는 것이다.

비난이 조직 문화에 자리 잡으면 이 자발적 진술이 사라진다. 사람은 처벌받을 걸 알면서 자기 실수를 상세히 보고하지 않는다. 대신 보고를 최소화하거나, 사실을 자기에게 유리하게 편집하거나, 아예 문제를 숨긴다. 이건 개인의 도덕성 문제가 아니라 합리적 선택이다. 정직하게 말했을 때 돌아오는 결과가 나쁘면, 정직하게 말하지 않는 게 이득이다.

문제는 이 침묵이 그 장애 한 건에서 끝나지 않는다는 점이다. 한 번 처벌받은 엔지니어는 다음번 이상 징후를 보고할 때도 주저한다. "이거 별거 아닐 수도 있는데 괜히 말했다가…"라는 생각이 먼저 든다. 조직 전체의 조기 경보 감도가 떨어진다. 진짜 큰 사고가 터지기 전에 나왔을

작은 신호들이 조용히 묻힌다.

구글의 사이트 신뢰성 엔지니어링 팀이 공개한 자료에서는 이를 "정보의 비대칭"이라고 부른다. 장애 당사자는 진짜 원인을 알지만, 처벌이 두려워 전체를 말하지 않는다. 조사자는 절반의 정보로 결론을 내린다. 그 결론은 절반만 맞는다. 다음 장애는 나머지 절반에서 온다.

1.2 심리적 안전은 느슨함이 아니다

여기서 자주 생기는 오해가 있다. "비난하지 않는다"를 "책임을 묻지 않는다"로 읽는 오해다. 심리적 안전(psychological safety)이라는 개념을 처음 접한 팀은 종종 이렇게 반응한다. "그럼 아무도 책임 안 지고 대충 하면 어떡해요?"

심리적 안전의 정의를 다시 보면 이 오해가 풀린다. 하버드 경영대학원의 에이미 에드먼드슨 교수가 정리한 개념에 따르면, 심리적 안전은 "팀 안에서 대인관계의 위험을 감수해도 안전하다는 공유된 믿음"이다. 여기서 핵심은 위험을 감수하는 것이지, 위험이 없다고 착각하는 게 아니다. 실수를 인정하는 것, 모른다고 말하는 것, 반대 의견을 내는 것이 처벌받지 않는다는 믿음이지, 결과에 대한 책임 자체가 사라진다는 뜻이 아니다.

실제로 심리적 안전이 높은 팀은 더 느슨하지 않다. 오히려 더 엄격한 기준을 유지하면서도 그 기준을 지키지 못했을 때 정직하게 보고할 수 있다. 반대로 심리적 안전이 낮은 팀은 기준 자체가 낮아진다. 문제를 발견해도 보고하는 순간 그 문제가 자기 것이 되므로, 발견하지 않은 척하는 게 개인에게 이득이기 때문이다.

blameless 포스트모템이 하는 일은 정확히 이 지점을 겨냥한다. 개인의 실수를 조사의 종착점이 아니라 출발점으로 삼는다. "누가 실수했나"에서 멈추지 않고 "왜 그 실수가 시스템을 뚫고 프로덕션까지 도달했나"로 넘어간다. 실수한 사람을 감싸기 위해서가 아니다. 실수는 어차피

일어난다. 사람은 피곤하고, 정보는 불완전하고, 압박은 존재한다. 시스템이 그 실수를 막아주지 못했다면, 그건 그 사람의 문제가 아니라 시스템의 설계 문제다.

1.3 숨긴 장애가 되돌아오는 세 가지 경로

장애를 비난하는 문화가 정착된 조직에서는 정보가 사라지는 대신 다른 형태로 재등장한다. 세 가지 경로가 흔하다.

첫째, 같은 원인이 다른 모습으로 다시 터진다. 근본원인을 제대로 조사하지 않고 표면적인 조치(재시작, 롤백, 수동 패치)로 마무리하면, 근본원인은 그대로 남는다. 몇 주 뒤 비슷하지만 조금 다른 조건에서 같은 원인이 다시 장애를 낸다. 조사팀은 "어, 이거 전에 본 것 같은데"라고 말하지만 기록이 없어서 확신하지 못한다.

둘째, 우회 지식이 특정 개인에게 갇힌다. 실제로 문제를 겪은 사람은 "아, 그 서버는 재부팅하면 안 돼요, 예전에 문제 있었어요"라는 걸 안다. 하지만 이 지식이 문서화되지 않으면 그 사람이 퇴사하거나 휴가를 가는 순간 조직은 그 지식을 잃는다. 다음 사람은 똑같은 실수를 처음부터 다시 겪는다.

셋째, 신뢰가 서서히 마모된다. 장애 보고가 축소되고 왜곡된다는 걸 경영진이나 고객이 감지하기 시작하면, 그 조직이 내놓는 다른 모든 보고도 의심받는다. "이번 장애는 영향이 적었습니다"라는 말을 믿을 수 없게 되면, 정상 운영 보고도 신뢰를 잃는다. 신뢰는 한 번 무너지면 회복에 훨씬 긴 시간이 걸린다.

1.4 포스트모템이 처벌 도구가 될 때

포스트모템이라는 형식 자체는 중립적이다. 이 형식을 처벌 도구로 쓸지 학습 도구로 쓸지는 조직이 그 문서를 어떻게 다루는지에 달렸다. 몇 가지 위험 신호가 있다.

먼저, 포스트모템 문서가 인사평가 자료로 흘러 들어가는 경우다. "이 사람이 몇 번의 장애에 이름이 등장했는가"를 성과 평가에 반영하는 순간, 그 조직의 포스트모템은 죽는다. 사람들은 자기 이름이 문서에 최대한 적게 등장하도록 문서를 편집하기 시작한다.

다음으로, 포스트모템 회의에서 "누가"를 반복해서 묻는 경우다. 진행자가 "그때 배포한 사람이 누구였죠"를 여러 번 물으면, 참석자들은 그 회의가 원인 조사가 아니라 책임자 색출이라는 걸 눈치챈다. 그 순간부터 방어적인 답변이 시작된다.

마지막으로, 관리자가 포스트모템 회의에 참석해 실시간으로 평가하는 티를 내는 경우다. 표정, 어조, 후속 질문의 방식에서 "이 사람 탓이다"라는 신호가 새어 나오면, 아무리 문서에 blameless라고 써 있어도 실제 문화는 그렇지 않다.

이 책의 다음 장들은 이 함정을 피하면서 실제로 작동하는 포스트모템을 만드는 구체적인 방법을 다룬다. 2장은 좋은 포스트모템 문서를 쓰는 구조를, 3장은 blameless 문화를 회의와 언어 수준에서 만드는 법을, 4장은 재발 방지 조치가 실제로 지켜지게 만드는 후속 관리를 다룬다. 이론이 아니라 다음 장애가 났을 때 바로 쓸 수 있는 틀을 목표로 한다.

2 읽히는 포스트모템 문서를 쓰는 법

포스트모템 문서를 처음 써보는 팀은 대개 두 가지 극단에 빠진다. 하나는 너무 짧다. "결제 서버가 죽어서 재시작했습니다. 끝." 이건 다음에 같은 일이 생겨도 아무 도움이 안 된다. 다른 하나는 너무 길다. 로그 전체를 복사 붙여넣기 하고, 관련 없는 배경 설명을 몇 페이지씩 붙인다. 아무도 끝까지 읽지 않는다. 좋은 포스트모템은 이 중간 지점에 있다. 읽는 사람이 5분 안에 무슨 일이 있었는지 파악하고, 필요하면 더 깊이 파고들 수 있는 구조다.

2.1 타임라인은 사실만, 판단은 뒤로

포스트모템의 첫 번째 섹션은 타임라인이어야 한다. 그리고 이 타임라인에는 판단이나 해석이 섞이면 안 된다. "14시 02분, 배포 파이프라인이 새 버전을 프로덕션에 반영했다"는 사실이다. "14시 02분, 담당자가 부주의하게 배포했다"는 판단이다. 이 둘을 섞으면 타임라인을 읽는 순간부터 독자는 방어적으로 읽거나 이미 내려진 결론을 그대로 받아들인다. 둘 다 학습에 방해가 된다.

타임라인을 쓸 때는 다음 순서를 지킨다. 먼저 무엇이 바뀌었는지(배포, 설정 변경, 트래픽 급증, 외부 서비스 장애)를 시각과 함께 적는다. 다음으로 언제 이상이 감지됐는지를 밝힌다. 자동 알림이었는지, 고객 문의였는지, 우연히 대시보드를 보던 엔지니어였는지 구분해 둔다. 이 구분이 나중에 "감지 시간을 어떻게 단축할까"라는 개선 논의의 재료가 된다. 마지막으로 대응 과정을 시각순으로 나열한다. 누가 무엇을 시도했고, 그 시도가 효과가 있었는지 없었는지까지 기록한다. 효과가 없었던 시도도 남긴다. 다음 사람이 같은 시행착오를 반복하지 않게 하기 위해서다.

타임라인 작성에서 흔히 놓치는 부분은 "왜 그 순간에 그 판단을

내렸는가"다. 사후에 보면 명백히 잘못된 판단도, 그 순간 가진 정보로는 합리적이었을 수 있다. 예를 들어 "롤백보다 핫픽스를 먼저 시도했다"는 결정이 결과적으로 30분을 더 끌었다고 해도, 그 순간 담당자는 "핫픽스가 더 빠를 것"이라는 근거 있는 판단을 했을 수 있다. 이 맥락을 남기지 않으면 나중에 이 문서를 읽는 사람은 "왜 이렇게 비효율적으로 대응했지"라고만 생각한다. 판단의 근거를 함께 적어야 그 판단 과정 자체를 개선할 수 있다.

2.2 5 Whys를 한 사람의 실수로 끝내지 않는 법

근본원인 분석에서 가장 널리 쓰이는 도구는 5 Whys다. "왜 장애가 났는가"를 다섯 번 정도 반복해서 물으며 표면적 증상에서 구조적 원인으로 내려가는 방법이다. 그런데 이 방법은 잘못 쓰면 결국 "그 사람이 실수했다"에서 멈추는 함정이 있다. 실제 예시로 이 함정과 탈출법을 살펴보자.

왜 결제가 실패했는가. 새 배포에서 널 포인터 예외가 발생했기 때문이다. 왜 그 예외가 발생했는가. 새 코드가 특정 필드가 항상 존재한다고 가정했기 때문이다. 왜 그런 가정을 했는가. 담당 엔지니어가 그 필드가 선택 값이라는 걸 몰랐기 때문이다. 여기서 멈추면 "담당자가 몰랐다"가 결론이 되고, 해결책은 "다음부터 조심하자"가 된다. 이건 재발을 막지 못한다. 더 물어야 한다.

왜 담당 엔지니어는 그 필드가 선택 값이라는 걸 몰랐는가. 스키마 문서에 그 정보가 없었기 때문이다. 왜 스키마 문서에 없었는가. 스키마와 문서가 자동으로 동기화되지 않고 수동으로 갱신되는 구조였기 때문이다. 여기까지 오면 결론이 완전히 달라진다. "그 엔지니어가 부주의했다"가 아니라 "스키마 문서를 수동 관리하는 시스템이 이런 실수를 구조적으로 유발한다"가 된다. 해결책도 달라진다. "다음부터 조심하자"가 아니라 "스키마에서 문서를 자동 생성하는 도구를 도입한다"가 된다. 전자는 다음 사람도 똑같이 겪을 조치고, 후자는 이 클래스의 문제 전체를 없애는

조치다.

5 Whys를 진행할 때 지켜야 할 원칙이 있다. 각 질문의 답이 "사람"이 아니라 "시스템, 프로세스, 도구"를 가리킬 때까지 계속 파고든다. "누가 실수했나"에서 멈추면 실패한 5 Whys고, "무엇이 이 실수를 가능하게 했나"까지 가면 성공한 5 Whys다. 다섯 번이라는 숫자에 집착할 필요는 없다. 세 번에 구조적 원인이 나올 수도 있고, 일곱 번이 필요할 수도 있다.

2.3 숫자로 남기는 임팩트와 대응 시간

포스트모템에는 반드시 정량적 지표가 들어가야 한다. 영향받은 사용자 수 또는 비율, 장애 지속 시간, 감지까지 걸린 시간, 감지부터 완전 복구까지 걸린 시간. 이 숫자들은 이 장애의 심각도를 객관적으로 보여줄 뿐 아니라, 시간이 지나면서 조직의 대응 역량이 얼마나 개선되는지 추적하는 근거가 된다.

아래는 포스트모템 요약에 흔히 들어가는 지표를 정리한 표다.

지표	의미	예시
감지 시간	발생부터 인지까지	12분
대응 시간	인지부터 착수까지	3분
복구 시간	착수부터 정상화까지	22분

이 세 숫자를 분리해서 기록하면 개선 대상이 명확해진다. 감지 시간이 길다면 모니터링과 알림 규칙을 개선해야 한다. 대응 시간이 길다면 온콜 프로세스나 에스컬레이션 체계를 손봐야 한다. 복구 시간이 길다면 롤백 자동화나 런북 정비가 필요하다. 이 셋을 뭉뚱그려 "장애가 40분 걸렸다"고만 적으면 어디를 고쳐야 할지 알 수 없다.

불확실한 수치는 추정으로 명시하고 근거를 남긴다. "영향받은 사용자는 약 3천 명으로 추정된다(에러 로그 카운트 기준, 재시도 성공 건 미제외)"처럼

어떤 데이터로 어떻게 계산했는지까지 적어두면 나중에 그 숫자를 신뢰할지 재검증할 근거가 남는다.

2.4 공개를 전제로 쓰는 문장 톤

포스트모템 문서를 쓰는 사람의 머릿속에는 항상 "이 문서가 팀 전체, 어쩌면 회사 전체, 경우에 따라 고객에게도 공개될 수 있다"는 전제가 있어야 한다. 이 전제는 내용의 정직성을 낮추기 위한 게 아니라, 표현 방식을 다듬기 위한 것이다.

구체적으로는 문장에서 특정 개인을 지목하는 표현을 피한다. "김OO 님이 실수로 프로덕션 설정을 변경했다" 대신 "배포 프로세스에서 스테이징과 프로덕션 설정 파일을 구분하는 안전장치가 없어, 프로덕션 설정이 의도치 않게 변경됐다"로 쓴다. 이 문장은 사실을 숨기지 않는다. 오히려 더 정확하다. 실수를 저지른 사람의 이름을 빼는 대신, 그 실수가 어떻게 가능했는지를 정확히 설명하기 때문이다.

또한 원인을 서술할 때 능동태보다 시스템 중심 서술을 쓴다. "엔지니어가 검증을 생략했다"보다 "검증 단계가 필수가 아닌 선택적 절차로 설계되어 있었다"가 더 유용하다. 전자는 사람의 행동을 지적하고 끝나지만, 후자는 그 행동이 왜 가능했는지 시스템 설계까지 드러낸다.

마지막으로, 초안 단계에서 당사자에게 먼저 보여주고 피드백을 받는 절차를 만든다. 문서가 공개되기 전에 당사자가 "이 부분은 제 기억과 다릅니다" 또는 "이 표현은 오해의 소지가 있습니다"라고 말할 기회를 주면, 문서의 정확도가 올라가고 당사자의 신뢰도 함께 쌓인다. 이 절차 자체가 blameless 문화를 문서 작성 과정 안에서 실천하는 방법이다.

3 blameless 문화는 규칙이 아니라 반복된 행동이다

"우리 팀은 blameless 문화입니다"라는 문장을 사내 위키에 써놓는다고 해서 그 문화가 만들어지지는 않는다. 문화는 선언이 아니라 반복된 행동의 축적이다. 특히 장애처럼 감정이 격해지기 쉬운 상황에서 리더와 진행자가 실제로 어떻게 행동하는지가 문서에 적힌 어떤 원칙보다 강하게 팀에 각인된다. 이 장은 그 행동을 구체적으로 다룬다.

3.1 리더가 회의에서 먼저 보여줘야 하는 반응

포스트모템 회의의 분위기는 회의가 시작되고 첫 3분 안에 결정된다. 그리고 그 3분을 결정하는 사람은 대개 그 방에서 가장 직급이 높은 사람이다. 리더가 팔짱을 끼고 "그래서 누가 이걸 배포했죠"로 회의를 시작하면 그 뒤로 어떤 blameless 원칙을 읊어도 소용없다. 참석자들은 이미 방어 태세에 들어갔다.

효과적인 리더는 회의를 다른 질문으로 시작한다. "먼저 대응해주신 분들 고생 많으셨습니다. 시작하기 전에, 이 회의의 목적은 무엇이 시스템을 뚫었는지 이해하는 겁니다. 누군가를 지목하려는 게 아닙니다"라는 말을 회의 초반에 명시적으로 언급한다. 당연한 말처럼 들리지만 이 말을 실제로 매번 반복해서 하는 리더는 드물다. 반복이 중요한 이유는 한 번 말했다고 팀이 그 원칙을 내재화하지 않기 때문이다. 특히 새로 합류한 팀원이나 과거에 다른 조직에서 비난 문화를 경험한 팀원에게는 이 반복이 필요하다.

리더가 취약성을 먼저 보여주는 것도 강력한 신호다. 자기 자신이 과거에 저지른 실수를 예로 들며 "저도 3년 전에 비슷한 실수로 장애를 냈습니다. 그때 배운 건…"이라고 말하는 리더 밑에서는 팀원들이 자기 실수를 훨씬

편하게 인정한다. 반대로 리더가 한 번도 자기 실수를 인정하지 않는 조직에서는 실수를 인정하는 게 곧 약점을 드러내는 행위로 여겨진다.

리더는 회의 중 누군가 방어적으로 반응할 때 그 반응도 부드럽게 다뤄야 한다. "제 잘못이 아니에요, 그때 문서에 그렇게 적혀 있었어요"라는 방어적 발언이 나왔을 때 리더가 "그게 아니라…"로 반박하면 방어전이 시작된다. 대신 "맞습니다, 문서가 그렇게 적혀 있었다는 게 핵심 문제네요. 그 문서는 왜 그렇게 적혀 있었을까요"로 받으면 방어를 인정하면서 동시에 논의를 시스템 쪽으로 자연스럽게 옮긴다.

3.2 회고 진행자의 질문 설계

포스트모템 회의를 진행하는 사람은 질문을 어떻게 던지느냐로 회의 전체의 방향을 조종한다. 좋은 진행자는 "누가"로 시작하는 질문을 거의 쓰지 않는다. 대신 "무엇이"와 "어떻게"로 시작하는 질문을 쓴다.

몇 가지 대비되는 예시를 보면 차이가 분명해진다. "누가 이 설정을 바꿨나요" 대신 "이 설정이 바뀐 경로를 추적할 수 있을까요"를 묻는다. "왜 테스트를 안 했나요" 대신 "이 변경이 테스트를 거치지 않고 배포될 수 있었던 경로는 무엇인가요"를 묻는다. "이걸 놓친 이유가 뭔가요" 대신 "이걸 발견할 수 있었던 지점이 있었다면 어디였을까요"를 묻는다.

이 재구성의 공통점은 주어를 사람에서 시스템, 프로세스, 경로로 바꾸는 것이다. 사람이 주어인 질문은 필연적으로 방어 반응을 부른다. 인간은 자신이 평가받는다고 느끼면 사고가 좁아지고 방어적으로 변한다. 시스템이 주어인 질문은 같은 사실 관계를 묻더라도 답하는 사람이 관찰자 입장에 설 수 있게 해준다. "제가 실수했어요"라고 고백하는 것과 "그 경로에 이런 허점이 있었어요"라고 설명하는 것은 심리적으로 완전히 다른 행위다.

진행자는 침묵을 견딜 줄도 알아야 한다. 어려운 질문을 던진 뒤 아무도

바로 답하지 않으면 그 침묵을 서둘러 메우려고 하지 않는다. 3초에서 5초 정도의 침묵은 사람들이 방어적인 첫 반응 대신 좀 더 신중한 답을 준비하는 시간이다. 진행자가 이 침묵을 못 참고 바로 다음 질문으로 넘어가거나 스스로 답을 제시하면 회의는 얕은 수준에서 끝난다.

3.3 비난 없이도 책임을 묻는 법

blameless 문화의 가장 흔한 오해는 "책임을 아예 묻지 않는다"는 것이다. 실제로는 정반대다. blameless 문화는 책임을 개인의 도덕적 결함이 아니라 역할과 시스템의 문제로 재정의함으로써 오히려 더 명확하게 책임 소재를 추적할 수 있게 한다.

구체적으로 세 층위로 책임을 나눠서 생각하면 도움이 된다. 가장 먼저 오는 층위는 즉각 대응의 책임이다. 이건 명확하다. 온콜 담당자는 알림에 응답할 책임이 있고 이 책임은 blameless 원칙과 무관하게 그대로 유지된다. 알림을 무시했다면 그건 시스템 문제가 아니라 온콜 프로세스를 지키지 않은 것이고 이 부분은 인사 절차로 다뤄질 수 있다.

두 번째 층위는 근본원인에 기여한 결정의 책임이다. 이 층위가 blameless 원칙이 가장 강하게 적용되는 지점이다. 특정 필드가 선택 값인 걸 몰랐던 엔지니어에게 "몰랐다"는 사실 자체를 책임으로 묻지 않는다. 대신 "이 정보를 알 수 있는 경로를 만들지 못했다"는 조직 차원의 책임을 묻는다.

마지막 층위는 재발 방지 조치를 실행하는 책임이다. 포스트모템에서 나온 액션 아이템은 누군가 담당자가 되어 실제로 완료해야 한다. 이 책임은 개인에게 명확히 배정되고 추적되고 완료 여부가 확인된다. 여기서는 blameless가 느슨함을 의미하지 않는다. 오히려 이 층위의 책임 추적이 느슨한 조직은 아무리 회의에서 좋은 대화를 나눠도 같은 장애를 반복한다.

이렇게 층위를 나누면 "책임을 묻지 않는다"는 오해가 풀린다. blameless는 첫 번째와 세 번째 층위의 책임은 그대로 유지하면서 두 번째 층위에서만

개인을 향한 비난을 시스템을 향한 질문으로 전환하는 것이다.

3.4 blameless가 무책임으로 변질되는 경계

마지막으로, blameless 원칙을 적용하는 과정에서 실제로 경계해야 할 실패 사례를 짚는다. 조직이 이 원칙을 처음 도입할 때 종종 반대 방향으로 과도하게 흔들린다.

첫 번째 실패 사례는 반복되는 실수를 계속 시스템 탓으로만 돌리는 경우다. 같은 사람이 세 번, 네 번 비슷한 유형의 실수를 반복한다면 이건 시스템 문제일 수도 있지만 그 개인의 숙련도나 업무 배치 문제일 수도 있다. 이 가능성을 완전히 배제하면 blameless는 방패가 아니라 무책임을 가리는 커튼이 된다. 반복 패턴이 보이면 별도의 일대일 대화로 다뤄야 한다. 다만 이 대화는 포스트모템 회의 자리가 아니라 별도의 자리에서 그 회의의 blameless 신뢰를 훼손하지 않는 방식으로 진행한다.

두 번째 실패 사례는 명백한 규정 위반을 시스템 문제로 뭉개는 경우다. 예를 들어 권한 없는 사람이 프로덕션 데이터베이스에 직접 접속해서 수동으로 데이터를 수정하다가 장애를 냈다면 “권한 체계가 허술했다”는 시스템적 원인 분석과 별개로 “허용되지 않은 접근을 시도한 것” 자체는 별도로 다뤄야 할 사안이다. 이 둘을 섞으면 규정 자체가 유명무실해진다.

마지막 실패 사례는 액션 아이템이 나왔는데 아무도 완료하지 않는 걸 방치하는 경우다. blameless가 “결과에 대한 책임도 없다”로 확장되면 회의는 매번 좋은 대화로 끝나지만 시스템은 하나도 개선되지 않는다. 다음 장에서 이 문제, 즉 재발 방지 조치가 실제로 실행되게 만드는 방법을 다룬다.

4 재발 방지가 실제로 작동하게 만드는 후속 조치 설계

포스트모템 회의가 아무리 좋은 분위기에서 진행되고 근본원인을 정확히 짚었다고 해도, 그 결과물인 액션 아이템이 실행되지 않으면 그 회의는 시간 낭비다. 오히려 나쁜 결과를 남긴다. 다음 장애가 났을 때 "저번에도 이거 고치기로 했잖아요"라는 말이 나오면, 포스트모템 자체가 신뢰를 잃는다. 이 장은 회의에서 나온 좋은 의도를 실제 완료까지 이어지게 만드는 구조를 다룬다.

4.1 액션 아이템이 죽는 이유

액션 아이템이 실행되지 않는 데는 몇 가지 반복되는 패턴이 있다. 이 패턴을 먼저 이해해야 그걸 막는 구조를 설계할 수 있다.

가장 흔한 패턴은 액션 아이템이 너무 추상적으로 적히는 것이다. "배포 프로세스를 개선한다"는 문장은 담당자가 봐도 무엇부터 해야 할지 모른다. 시작점이 불분명한 작업은 다른 급한 일에 밀려서 무한정 뒤로 미뤄진다. 좋은 액션 아이템은 "무엇을, 어떤 완료 기준으로"까지 구체적이어야 한다. "배포 파이프라인에 스테이징 테스트 통과를 필수 게이트로 추가한다. 완료 기준은 이 게이트를 우회하는 배포가 CI에서 자동으로 막히는 것"처럼 적으면, 담당자가 시작 지점을 바로 안다.

두 번째 패턴은 담당자가 명확하지 않은 경우다. "팀에서 검토한다"처럼 특정 팀 전체가 담당으로 지정되면, 그 안에서 아무도 자기 일이라고 생각하지 않는다. 모두의 일은 아무의 일도 아니라는 말이 정확히 여기 적용된다. 액션 아이템에는 반드시 이름이 하나 배정돼야 한다. 여러 사람이 필요한 작업이라면 그중 한 명이 완료 여부의 최종 책임을 진다.

세 번째는 우선순위가 아예 없다는 점이다. 포스트모템 회의에서 나온 액션 아이템 열 개가 모두 같은 우선순위로 백로그에 던져지면, 평상시 기능 개발 업무에 밀려서 대부분 실행되지 않는다. 특히 이 아이템들이 눈에 보이는 사용자 가치를 만들지 않는 인프라성 작업일수록 더 그렇다.

네 번째는 추적할 방법 자체가 없다는 것이다. 회의가 끝나고 액션 아이템 목록이 문서 어딘가에 저장된 채로 다시는 열어보지 않으면, 완료됐는지 아닌지 아무도 모른다. 완료 여부를 확인하는 별도의 절차가 없으면, 그 목록은 사실상 존재하지 않는 것과 같다.

4.2 회의 안에서 확정하는 법

액션 아이템의 운명은 대부분 회의가 끝나는 순간에 이미 결정된다. 회의 이후에 "누가 이거 할지 정리해서 알려주세요"로 마무리하면, 그 정리는 대개 이뤄지지 않거나 며칠 뒤 흐지부지된다. 회의 안에서 다음 세 가지를 확정하고 끝내야 한다.

첫째, 각 액션 아이템마다 담당자 이름을 그 자리에서 정한다. "이거 누가 하실 수 있을까요"라고 물어서 그 회의에 참석한 사람 중 한 명이 손을 든다. 담당자가 그 자리에 없다면, 회의 종료 전에 그 사람에게 확인받는 절차를 명시한다.

둘째, 완료 기한을 정한다. 막연한 "빠른 시일 내"가 아니라 구체적인 날짜다. 큰 작업이라면 전체 완료 기한과 함께 중간 체크포인트 날짜도 정한다.

셋째, 우선순위를 정한다. 아래 표처럼 심각도에 따라 세 단계로 나누면 충분하다.

등급	기준	처리
긴급	같은 장애 재발 가능	이번 스프린트
중요	재발 위험 낮춤	다음 스프린트

등급	기준	처리
개선	전반적 품질 향상	백로그 등재

긴급 등급 아이템은 다른 기능 개발 작업보다 우선한다는 원칙을 조직 차원에서 미리 합의해둔다. 이 합의가 없으면 매번 “이번엔 기능 출시가 급해서…”라는 이유로 밀린다.

4.3 일회성 조치를 조직의 학습 자산으로

포스트모템에서 나온 조치를 그 장애 하나를 막는 데 그치지 않고, 조직 전체가 배우는 자산으로 확장하는 방법이 있다. 핵심은 개별 장애의 교훈을 패턴으로 추출하고, 그 패턴을 다음 설계 단계에 미리 반영하는 것이다.

가장 간단한 방법은 포스트모템 문서 전체를 검색 가능한 저장소에 모아두고 정기적으로 검토하는 것이다. 분기마다 지난 포스트모템들을 모아 놓고 “반복되는 근본원인 카테고리가 있는가”를 살펴본다. 예를 들어 지난 세 건의 장애가 모두 “설정 변경이 검증 없이 프로덕션에 반영됨”이라는 패턴을 공유한다면, 이건 개별 사건이 아니라 조직 차원의 구조적 취약점이다. 이런 패턴이 발견되면 개별 액션 아이템 수준을 넘어, 설정 변경 파이프라인 전체를 재설계하는 프로젝트처럼 더 큰 단위의 투자로 격상시킨다.

두 번째 방법은 신규 설계 리뷰나 아키텍처 검토 과정에 “과거 포스트모템 체크리스트”를 넣는 일이다. 새로운 서비스나 기능을 설계할 때 “이 설계가 과거에 발생한 장애 패턴 중 어느 것을 반복할 위험이 있는가”를 명시적으로 검토하는 단계를 넣는다. 이렇게 하면 포스트모템이 과거를 기록하는 데서 끝나지 않고 미래의 설계 결정에 영향을 미친다.

세 번째 방법은 신규 입사자 온보딩에 주요 포스트모템 사례를 포함시키는 데 있다. 조직이 과거에 어떤 실패를 겪었고 그 실패에서 무엇을 배웠는지를

신규 입사자가 초기에 접하면, 그 조직이 왜 지금의 프로세스와 안전장치를 갖추고 있는지 맥락을 이해하게 된다. 이 맥락 없이 규칙만 전달받으면, 신규 입사자는 그 규칙을 관료적 절차로만 받아들이고 우회하려는 유혹을 느낀다.

4.4 포스트모템 데이터베이스를 다음 설계로 연결하기

가장 성숙한 조직은 포스트모템을 단순한 기록 문서가 아니라 하나의 데이터베이스로 다룬다. 각 포스트모템에 원인 카테고리, 영향받은 시스템, 심각도, 감지 방식 같은 메타데이터를 태그로 남기면, 시간이 지나면서 이 데이터가 쌓여 정량적인 분석이 가능해진다.

예를 들어 "지난 1년간 발생한 장애 중 몇 퍼센트가 배포 직후 한 시간 이내에 발생했는가"를 확인하면, 배포 직후 모니터링을 강화하는 투자의 우선순위를 데이터로 뒷받침할 수 있다. "감지 시간이 가장 긴 원인 카테고리는 무엇인가"를 확인하면, 모니터링 개선 투자를 어디에 집중할지 판단할 수 있다. 이런 분석은 개별 포스트모템 하나만 봐서는 나오지 않는다. 누적된 데이터가 있어야 나온다.

이 데이터베이스화 작업은 처음부터 완벽할 필요는 없다. 작게 시작해도 된다. 포스트모템 문서 상단에 몇 개의 표준화된 태그 필드만 추가하는 것부터 시작할 수 있다. 원인 카테고리(코드 결함, 설정 오류, 용량 부족, 외부 의존성 장애, 인적 절차), 심각도 등급, 감지 방식 정도면 충분하다. 이 필드들이 몇 분기 쌓이면, 그 조직만의 실패 지문이 드러난다. 그 지문이 다음 분기의 안정성 투자 우선순위를 결정하는 가장 신뢰할 수 있는 근거가 된다.

결국 포스트모템 문화가 성숙한다는 것은 매번 새로운 장애를 처음 겪는 것처럼 대응하는 조직에서, 과거의 실패를 미래의 설계에 미리 반영하는 조직으로 옮겨간다는 뜻이다. 이 책에서 다룬 네 가지, 즉 비난 없는 정보 흐름, 읽히는 문서, 반복된 행동으로서의 blameless 문화, 그리고 실행되는

후속 조치는 서로 분리된 기술이 아니라 한 시스템의 네 부분이다. 하나라도 빠지면 나머지 셋도 온전히 작동하지 않는다. 다음 장애가 났을 때 이 넷을 한 번에 완벽하게 갖추려 하지 말고, 지금 팀에 가장 약한 한 부분부터 고쳐 나가는 것이 실질적인 출발점이다.