1장. 살아남기 위한 방법
소프트웨어 프로젝트 생존 훈련에 참가하게 된 여러분을 환영한다. 소프트웨어 프로젝트에서 생존할 가능성이 매우 낮기는 하지만 사실 꼭 그렇게 낮아야 할 필요는 없다. 소프트웨어 프로젝트에서 살아남으려면 우선 문명화된 방식(civilized way)으로 프로젝트를 시작하면 된다. 그러면 출발점부터 단순히 생존하는 것 이상이 가능할 것이다.

2장. 소프트웨어 프로젝트의 생존 테스트
간단한 테스트로 소프트웨어 프로젝트의 건강 상태를 평가할 수 있다. 만약 테스트 결과가 적신호라면 점수를 올리기 위한 조치를 취하여 프로젝트 상태를 개선시킬 수 있다.

3장. 생존의 개념
잘 정의된 개발 프로세스는 소프트웨어 프로젝트의 생존에 중요하고, 필수적인 요소다. 이것은 소프트웨어 개발 관련자들이 쓸데없는 데 신경 쓰지 않고 생산적인 업무에 전념하여 프로젝트를 안정적으로 끝낼 수 있도록 한다. 빈약한 프로세스(poorly planned process)는 개발자들이 실수를 수정하는 데 많은 시간을 소비하도록 한다. 프로젝트에서 성공하기 위한 수단 대부분은 상류(upstream) 활동에 있다. 식견 있는 소프트웨어 이해 관계자들은 하류의 문제를 최소화하기 위해 상류 활동에 역량을 집중한다.

4장. 생존 기술
소프트웨어 프로젝트는 본질적으로 복잡한 것이다. 복잡한데다 계획까지 엉성하다면 절대로 성공할 수 없다. 계획 수립이 잘 된 프로젝트는 통제도 잘 되며 진척 상황도 눈에 보인다. 팀원들이 업무를 잘 수행하도록 지원도 잘 된다. 또한 소프트웨어 프로젝트는 본질적으로 위험한 것이다. 적극적으로 리스크를 관리하지 않으면 성공하기 힘들다. 리스크 관리를 잘 하려면 프로젝트 초반부터 사용자를 계속 참여시켜야 한다. 그리고 제품에 들어가는 기능은 최소한으로 유지시키도록 노력해야 한다. 원하는 결과가 나올 수 있게 해야 함은 말할 것도 없다.

5장. 성공적인 프로젝트란
소프트웨어 프로젝트란 뭔가를 찾아내고 발명하는 프로세스다. 프로젝트 계획을 세울 때 좋은 방법 중 하나는 '단계별 납품(staged delivery)' 방식을 적용하는 것이다. 단계별 납품이란 소프트웨어 기능을 단계별로 개발하고 납품하는 것을 말한다. 물론 이때 중요한 기능을 먼저 완성해야 한다. 프로젝트를 수행할 때 많은 활동들이 부분적으로 겹쳐져 진행되며, 이 활동들은 추상적인 것으로부터 점차 구체적인 것으로 옮겨간다. 프로젝트 진행 시 코딩량(source code)은 직선이 아닌 S곡선을 그린다. 그리고 코딩은 대부분 프로젝트의 중반(middle third)에서 하게 된다. 코딩량을 관찰하면 프로젝트 상태를 알 수 있다. 상급 관리자, 고객, 사용자는 마일스톤 별 산출물의 수준을 보고 프로젝트가 잘 진행되는지 여부를 파악하게 된다.

6장. 움직이는 표적 맞추기

효율적인 프로젝트는 변경을 통제하지만, 비효율적인 프로젝트는 변경이 프로젝트를 통제한다. 성공적으로 변경을 통제하기 위해서는 “변경위원회(CCB)"를 설립하여, 사전에 정해진 룰에 따라 변경을 제한하고, 주요 산출물에 대한 변경을 통제해야 한다.

7장. 사전 계획
성공하는 프로젝트는 계획 수립을 일찍 한다. 사전(preliminary) 계획을 수립할 때는 프로젝트 비전을 정의하고, 최고 의사결정권자(executive sponsor)가 누구인지 찾아내야 한다. 또한 과업 범위에 대한 목표 설정, 리스크 관리, 효율적인 인력 운용 등에 대한 전략도 세워야 한다. 소프트웨어 개발 계획서에 이러한 사전 계획을 포함시켜야 한다.

8장. 요구사항 개발
요구사항을 개발하는 동안, 사용자 인터페이스 프로토타입과 사용자 매뉴얼과 요구 명세서를 여러 버전으로 작성하면 소프트웨어 개념이 더욱 명확해진다. 이 방법을 통해 요구사항의 최적 집합을 구할 수 있으며, 훌륭한 아키텍처를 만드는 데 기초를 다진다. 또한, 시간이 많이 소요되는 상세 요구사항 문서를 없앰으로써 프로젝트를 간소화시키며, 사용자 문서 때문에 중요작업(critical path)이 영향을 받지 않게 한다.

9장. 품질 보증
예전에는 소프트웨어 품질 보증을 전적으로 테스트 단계에서 하는 것으로 생각했다. 그러나 효율적인 프로젝트에서의 품질 보증은 이제 테스트, 테크니컬 리뷰, 프로젝트 계획 등과 같이 프로젝트 초기에 경제적인 방법으로 결함을 발견하여 수정하기 위한 모든 방법을 포함한다.

10장. 아키텍처
소프트웨어 아키텍처는 프로젝트에 대한 기술 구조를 제공한다. 좋은 아키텍처는 프로젝트를 안정적으로 수행할 수 있도록 하지만, 그렇지 못한 아키텍처는 프로젝트 수행을 거의 불가능하게 만든다. 좋은 소프트웨어 아키텍처 문서에는 전반적인 프로그램 구조, 발생 가능성이 높은 변경사항에 대한 아키텍처의 대응 방안, 다른 시스템에서 이미 개발되었거나 사서 쓸 수 있는 컴포넌트가 기술되어 있다. 또한 표준적인 기능, 요소들에 대한 설계 방안이 제시되어 있다. 그리고 각 시스템의 요구사항에 대하여 아키텍처를 어떻게 수립할지 열거함으로써 하류에서 발생될 수 있는 잠재적인 비용을 줄여준다.

11장. 최종 준비
최종 준비 시기는 요구사항 개발과 아키텍처 이전에 수행 되었던 사전 계획을 확장하고 수립한다. 최종 준비 시기가 되면 프로젝트 팀은 프로젝트의 첫째 추정 작업을 할 준비를 하며, 가장 중요한 기능을 우선해서 납품하기 위한 계획을 수립한다. 그리고 기타 계획을 재조정한다.

12장. 단계 계획 수립의 시작
단계별 계획을 시작할 때는 해당 단계에서 수행될 작업의 상세한 과정에 대한 단계 시작 시의 계획을 수립한다. 프로젝트 팀은 해당 단계의 상세설계, 코딩, 테크니컬 리뷰, 테스트, 통합 및

기타 작업을 수행하는 방법에 관한 개별 단계 계획을 수립한다. 이때 가장 많은 노력이 필요한 작업은 해당 단계의 프로젝트 진척도를 추적하기 위한 상세 마일스톤 목록을 작성하는 것이다. 이러한 마일스톤을 생성하려면 많은 수고를 해야 하지만, 이를 통해 프로젝트 상태를 쉽게 파악할 수 있고 리스크를 줄일 수 있으므로 그만한 가치가 있다.

13장. 상세설계

각 단계에서 상세설계는 아키텍처 설계를 확장하는 것으로, 아키텍처 설계와 동일한 주제를 다루지만 이들을 보다 상세하게 처리하는 것이다. 얼마나 상세하게 설계할지는 해당 프로젝트 규모와 개발자의 기술 수준(expertise)에 따라 다르다. 상세설계를 리뷰하면 프로젝트의 품질과 비용 측면에서 상당한 효과를 얻을 수 있다. 프로젝트 제1단계에 대한 상세설계는 아키텍처의 품질을 검증하는 것과 같은 특수한 작업이 필요할 때도 있다.

14장. 구축

구축은 개발팀이 소프트웨어에 생명을 불어 넣는 흥미로운 시간이다. 구축 이전의 작업들이 효과적으로 완수되었다면 구축은 조화롭고 생산적인 시간이 되어, 개발자는 꾸준히 시스템에 기능을 추가하고 일별 빌드(dairy build) 및 스모크 테스트를 수행할 수 있을 것이다. 성공적인 프로젝트 팀은 구축 단계에서 소프트웨어를 더 간결하게 만들고, 소프트웨어에 가해진 변경을 통제하는 방법을 찾는데 더 많은 주의를 기울인다. 프로젝트 관리자가 상세 마일스톤, 결함, 10대 리스크 목록, 시스템 등을 포함하는 핵심 진척 지표들을 살펴보면 모든 진척 사항이 한눈에 파악된다.

15장. 시스템 테스트

시스템 테스트는 구축과 병행하거나 혹은 절반을 남겨 놓고 실시한다. 이때 시스템을 처음부터 끝까지, 그리고 여태껏 개발자가 수정한 결함을 노출시켜가면서 테스트한다. 테스트 담당자는 새로운 코드를 통합 시킬 수 있을 만큼 시스템의 품질이 높다는 것을 보증함으로써 개발자를 지원한다. 또 개발자는 보고된 결함을 신속히 수정하여 테스트 담당자를 지원한다.

16장. 소프트웨어 릴리스

각 단계를 종료할 때마다 소프트웨어를 릴리스 가능한 상태로 만들어두는 것이 통합 실패나 품질 저하 리스크를 관리하는 필수적인 방법이다. 소프트웨어가 릴리스할 만한 상태인지를 직관적으로 결정하기는 어렵다. 그러나 다행히도 몇 가지 간단한 통계적 기법을 사용하면 이런 결정에 도움을 받을 수 있다. 릴리스 단계는 상당히 분주한 시간이며, 릴리스 체크리스트를 활용하면 여러 문제를 피할 수 있다.

17장. 단계 마감

우리는 각 단계의 마감을, 진행 과정을 수정하고 해당 단계로부터 경험적 지식을 얻기 위한 기회로 삼을 수 있다. 프로젝트가 진행될수록 보다 정확한 추정 작업을 할 수 있으며, 이 추정은 다음 단계 시작을 위한 계획 수립시 탄탄한 기초를 제공한다. 프로젝트를 진행하면서

얻은 현재까지의 모든 경험은 이후 프로젝트에서 활용할 수 있도록 소프트웨어 프로젝트 로그에 기록해야 한다.

18장. 프로젝트 이력

소프트웨어 프로젝트 이력(software project history) 문서에 저장되어 있는 정보는 향후 프로젝트에 유용하게 쓰일 것이다. 소프트웨어 프로젝트 이력은 각 단계가 끝날 때마다 업데이트하는 소프트웨어 프로젝트 로그(software project log) 데이터를 사용하며, 이 정보에서 일반적인 교훈을 추출해낸다. 프로젝트가 제대로 진행되었다면, 소프트웨어 프로젝트 이력에 필요한 대부분의 데이터는 언제든지 사용이 가능하므로 프로젝트 이력을 만들기도 용이하다.

19장. 생존 안내서

이 책은 주요 가이드라인을, 세계에서 가장 효율적인 소프트웨어 개발 조직 가운데 하나인 NASA 소프트웨어공학연구소(Software Engineer-ing laboratory)의 가이드라인과 함께 제시하고 있다. 이 장의 마지막 부분에는 추가로 읽어볼 수 있는 서적 목록과 기타 자료를 소개한다.