開発私自身の経験では、明確で表現力豊かな特定のセキュリティ要件を持つ開発者は、非常にタイトなマシンを構築できます。  
–リック・スミスファッションの奴隷になると、アメリカのマネージャーは思春期の少女を険しい個性主義者のように見せます。  
– Archilochus  
これまで、さまざまなセキュリティアプリケーション、テクノロジー、および懸念事項について説明してきました。  
これにより、リスク分析、システムエンジニアリングの方法論、そして最後に、安全なコードを作成するためのチームの管理方法などのトピックがもたらされます。  
1980年代のオレンジブックから現在のアジャイル開発にいたるまで、多くの人々が1つがあると主張し、その瞬間の情熱に対して宗教的な熱意を得ています。  
1960年代、フレッドブルックスはチームを率いて、世界で最初の本当に大規模なソフトウェアプロジェクトであるIBM S / 360メインフレームのオペレーティングシステムを開発しました。  
本質的に難しい仕事を容易にする魔法の公式はありません。  
安全な開発の管理は、ヘッジホッグの知識ではなく、フォックスの知識です。  
2017年に、ケンブリッジで学部生に教える方法を変更しました。  
しかし、実際のシステムのほとんどは両方を必要とし、それらは複雑な方法で絡み合っています。  
どちらも、事故によるものであれ、氷の結果によるものであれ、何がうまくいかないのかについての体系的な思考を伴います。  
このコースは、2019年にサバティカルで働いていたときに同僚のAlastair Beresfordによってさらに開発されました。ソフトウェアとセキュリティエンジニアリングに関する2020年のコースは、パンデミック[89]のおかげで、10のビデオ講義としてオンラインになりました。  
この本のように、プロトコルや暗号の基本から定義まで、基本的なことを紹介し、次に、人と組織の問題の重要性と、技術的な問題の重要性を、ケース履歴で示します。  
成功とは、態度や仕事のやり方、スキルです。  
」と「私たちはそれを正しく構築していますか？  
次に、組織の行動の結果として発生する危険のいくつかについて説明します。これは、実際には無視されがちな内部関係の脅威です。  
27.2リスク管理  
リスク管理は、企業または実際に国に対するすべてのリスクを管理するより広範なフレームワーク内で行う必要があります。  
コロナウイルスの危機により、パンデミックは英国を含む多くの国のリスク記録のトップにありましたが、ほとんどの政府は、リストラのいくつかの場所にあったテロへのレジリエンス予算のはるかに多くを費やしたことを誰もが明らかにしたはずです。  
英国はこの病気が深刻になる前に2か月を浪費し、数万人の命が犠牲になりました。  
 私の大学の統治機関が使用している一般的な方法は、問題が発生する可能性のあるもののリストを作成し、深刻度と発生確率について1から5のスコアを付け、これらを掛け合わせて数を得るというものです。 1から25の間。  
次に、これらの各リスクを軽減するために講じる措置を書き留め、リスク委員会で、各リスクをどの程度軽減するかについて議論します。  
次に、すべてのリスクを順番にランク付けし、1人の上級役員を各リスクの所有者に割り当てます。  
 それが殺すかもしれない何人の人々によって（何百万？  
 何ダース？  
 そして、各世紀にどれだけの数を期待するかによって確率を評価します。  
 可能性については4、2017年には次のように述べています。「英国では、5年ごとに1つ以上の重大な危険が発生することが予想されます。  
次に、パンデミックのための検疫計画やPPE備蓄、洪水や地震による被害を制限するための建築基準法やゾーニングなど、緩和策によって合理的に実用的なものを検討します。  
さまざまな方法で問題が発生する可能性があります。  
テロについてはすでにセクション26.3で説明しました。ここでは、失敗のもう1つの側面がポリシーのオーバーシュートであったことを補足しておきます。  
ビジネスでも、政治的行動と組織的行動の両方が合理的なリスク管理の邪魔になっていることがわかります。  
標準的な方法は、年間損失期待値（ALE）を計算することです  
銀行のITシステムの一般的なALE分析には、図27.1に示すような項目を含め、数百のエントリがある場合があります。  
、大規模な送金詐欺などの低確率でリスクの高い損失の発生率は、ほとんど当て推量ですが、保険の見積もりを依頼することで大まかな健全性チェックを取得できる場合があります。英国政府は、CRAMMと呼ばれるツールを損失タイプAmount Incidence ALE SWIFT詐欺に使用します$ 50,000,000 .005 $ 250,000 ATM詐欺（大規模）  
 $ 20,000 .5 $ 10,000 Tellerが現金を受け取る$ 3,240 $ 648,000図27.1：–年間損失期待値（ALE）の項目  
しかし、このような確率の低い脅威のテーブルを作成するプロセスは、反復的な推測に過ぎない傾向があります。  
その後、CISOが政治的に可能であるとクライアントが言った最大のセキュリティ予算を正当化するものまで、合計を微調整します。  
ポイントは、ALEはある程度価値があるかもしれませんが、どの部分がデータに基づいているか、どの部分が当て推量にあり、どの部分がオフィス政治にあるかを理解する必要があります。  
産業の発展はその方法と規制の歴史により、産業によって異なります。  
ワシントンはブリュッセルより安全性を重視しておらず、OEMが複数の製品を所有するよりもEUの安全仕様に合わせて設計する方が簡単であるため、最大の市場であるEUはますます世界の安全規制機関になりつつあります。  
現在の目的のために、自動車、飛行機、医療機器のソフトウェアは、承認された手順に従って開発され、後で説明する分析を受け、特定の方法でテストされる必要があります。  
しかし、保険事業も完全に科学的ではありません。  
しかし、保険は循環型業界であり、2017年頃から、多くの新しい企業がサイバー犯罪に対する保険の提供を開始し、市場から利益を絞り出しています。  
そのため、リスクを評価するほとんどの保険会社の能力は現在制限されています。彼らが何をするかのメカニズムについては、セクション28.2.9でさらに説明します。  
（コロナウイルスの危機は、一部の保険会社がビジネス中断リスクポリシーへの支払いを拒否するため、相関リスクについて企業に教えています。保険のポイントは何ですか。）  
取り組まれているリスクは表面的には運用上に見えるかもしれませんが、実際には法律、規制、PRのリスクです。  
したがって、過失の主張は業界または職業の現在の基準によって評価され、群れを追跡する強い動機を与えます。  
。  
1980年代半ば、ハッカーについて誰もが話しました（たとえ数が少なくても）  
80年代後半から、ウイルスが企業の想像力を乗っ取り、人々はアンチウイルスソフトウェアを販売する金持ちになりました。  
1990年代後半には、PKIが熱狂しました。  
このすべてのフープラの中で、セキュリティ専門家はレベルの頭を維持し、真の脅威が何であるかを理解するために努力しなければなりません。  
まず、安全工学から学べることを見てみましょう。  
27.3安全が重要なシステムからの教訓  
障害のクラスに応じて、安全性、ビジネス、セキュリティ、環境のいずれかが重要になります。  
この問題に関する大規模な文献があり、リスクをインテリジェントに管理するのに役立つ多くの方法論が開発されています。  
27.3.1安全工学の方法論  
通常の手順は、危険を特定してリスクを評価することです。それらに対処するための戦略を決定する（回避、制約、冗長性...）  
テストの結果は、実際に実行できるシステムだけでなく、実行を正当化するための安全ケースの統合部分です。  
これは、道路車両用のISO 26262など、特定の業界向けのより専門的な規格に拡張されています。  
ffi（b）  
通常、ハザード分析、安全機能と完全性の要件、およびテストの結果（コンポーネントレベルとシステムレベルの両方）で構成されます。  
テストは、認定された第三者が行う必要がある場合があります。自動車では、企業は、独立した管理の下で、同じ会社の別の部門によって行われた安全ケースを回避します。  
上部にはブランドがあり、そのバッジは車の前部にあります。  
 車の場合、通常は同じ会社ですが、常にそうとは限りません。他の業界では、ブランドとOEMはまったく別のものです。  
自動車業界では、ブランドは車を型式承認し、主要な責任を負いますが、問題が発生した場合に備えてコンポーネントサプライヤーに補償を要求します（ほとんどの国の法律では、死傷の責任を放棄することは許可されていません）。ブランドは、安全機能と完全性の重要な部分、したがって安全ケースをサプライチェーンに依存しています。  
次に、一般的な安全工学の手法と、それらが教えてくれることを見てみましょう。  
27.3.2ハザード分析  
例として、図27.2のモーター逆転回路を考えます。  
ただし、これには潜在的な問題があります。スイッチの2つの極のうち1つだけが動くと、バッテリーが短絡し、火災が発生する可能性があります。  
ここで、スイッチの故障はモーターではなくバッテリーではありません。  
ハザード除去は、セキュリティエンジニアリングにも役立ちます。  
一般に、信頼できるコンピューティングベースを最小化することは、危険をなくすための練習です。  
たとえば、伝染病で誰が誰に感染したかを監視する連絡先追跡アプリを設計している場合、1つのアプローチは、すべての人の携帯電話の位置履歴の中央データベースを持つことです。  
次に、プライバシーの向上とトレースの向上のどちらかを選択するというポリシー決定があります。  
27.3.3フォールトツリーと脅威ツリー  
失敗する可能性のあるものを特定する一般的なトップダウンの方法は、ルートが望ましくない動作であり、その連続するノードがその考えられる原因であるツリーが構築されるフォールトツリー分析です。  
それは自然な方法でセキュリティエンジニアリングに引き継がれます。  
 現金自動預け払い機からの詐欺。  
あなたはそれぞれの望ましくない結果から始めて、考えられる直接的な原因をそれぞれ書き留めることによって逆方向に働きます。  
ツリーの葉を一周することで、セキュリティポリシーを破る可能性のある技術的な攻撃、運用上の誤解、物理的な侵入などの各組み合わせを確認できるはずです。  
一部の亜種では、攻撃ブランチに対策サブブランチがあり、対策ブランチが異なる色で強調されている場合があります。  
  
安全重視の世界に戻ると、ハザード分析を行う別の方法は、故障モードと影響分析（FMEA）です。  
これには、ミッションに影響が及ぶまで、システムの各コンポーネントの障害による影響を追跡することが含まれます。  
たとえば、エンジンが故障した場合に、空港や飛行機に乗ることができない海または山を越えて飛行機を飛行させる場合、エンジンの出力は非常に重要です。  
あなたは、単一エンジンの航空機が信頼できるエンジンを使用することを主張し、メンテナンススケジュールを調整します。飛行機には複数の燃料タンクがあります。  
重大であることが判明した故障モードを失った人々の航空宇宙の例は、スペースシャトルチャレンジャーの1986年の損失です。  
寒さによりもろくなったOリングが故障し、シャトルと7人の乗組員が失われた。  
これは、失敗はしばしば技術的なものだけでなく、組織内での人々の行動にも関係していることを示しています。たとえば、自動車のように保護メカニズムが制度の境界を越える場合、工学だけでなく法律と経済学についても考える必要があります。  
  
フォールトツリーとFMEAはどちらも、システムを本当によく理解しているアナリストに依存しています。それらは自動化が難しく、完全に再現可能ではなく、サブシステムに微妙な変更を加えることで逆転する可能性があります。  
人々が時間をかけて学んだアプリケーションに。  
自動車の安全性では、サプライチェーンが複雑なため、車両とそのサブシステムの複数の連動分析を行う必要があります。  
2つ以上のランプを使用することでランプ障害のリスクを軽減するだけでなく、スイッチの障害を心配し、スイッチが電子的になると、ハードウェアおよびソフトウェアの障害の可能性がある障害ツリーを構築します。  
この分析は、キャビンCANバスとパワートレインCANバスの間にファイアウォールを設けるという設計上の決定につながる可能性があります。  
 より一般的には、安全からセキュリティへの移行は、内部的には体系的に考える必要があることを意味します。  
では、悪意のある内部関係者を脅威モデルに組み込むにはどうすればよいでしょうか。  
より複雑なシステムについては、2003年にWindowsとOfficeをより安全にするための大きな推進に続いてMicrosoftが採用した方法論が、Frank SwiderskiとWindow Snyder [1851]によって説明されています。  
基本的な考え方は、保護しようとしている資産（トランザクションを実行する能力、機密データへのアクセスなど）だけでなく、。  
あなたは信頼レベルが何であるかを考え出そうとします。障壁があるところ;また、特定の障壁を克服するために、なりすまし、改ざん、否認、情報開示、サービス拒否、特権の昇格など、どのような手法を使用できるか。  
大量のデータを管理するには、さまざまな方法があります。  
したがって、問題のハザードが存在する場合、事故を引き起こすには少なくとも2つの故障が必要であることをグラフィカルに示すことができます。  
、ハザードから始めて、トップダウンプロセスで制御を設計し、システムのアーキテクチャ設計につながります。これは、相互作用する制御ループを切り離すのに役立ちます[1150]。  
複雑さを管理しやすくするために、安全とセキュリティの目標の階層を編成する必要がある場合があります。  
この階層は、業界で使用されている用語に応じて、リスクマトリックスまたはリスク処理計画を推進できます。  
27.3.6リスクの定量化  
コンポーネントの故障率は統計的に測定できます。ソフトウェアのバグの数は、次の章で説明する手法で追跡できます。また、さまざまなタイプのアクティビティでオペレータエラーが発生する可能性については、多くの経験があります。  
タスクのエラー率は、その親しみやすさと複雑さ、プレッシャーの量、成功への手がかりの数によって異なります。  
ただし、論理的な思考が必要であり、オペレーターがプレッシャーを受けている混乱しやすい環境で初めてタスクを実行する場合は、正常に完了できない可能性があります。  
同じ教訓が、ある航空事故調査から次々と出てきた。  
指針となる原則の1つは、デフォルトで安全な状態にすることです。つまり、核反応を弱めること、航空機をまっすぐ水平に戻すこと、または自律車両を道路脇で停止させることです。  
芝生の縁があると、車両は道路のどちら側にあるのかわからないことがあります。ボーイング737Maxのクラッシュ（セクション28.2.4で詳しく説明します）  
緊急時の安全性の使いやすさのもう1つの原則は、オペレーターに提供される情報と、オペレーターが使用できるコントロールをシンプルかつ直感的に保つことです。  
今日の誘惑は、可能であればオペレーターにすべてを与えることです。  
今日では、2つではなく50の警報があり、パイロットは電子飛行情報システムのどのメニューのどの画面を見ればよいのかを考えるのに苦労しています。  
海軍の例は、UIの混乱が主な要因であったシンガポール海峡でのUSSマケインの2017年衝突です。  
したがって、完全に自律的ではないシステムは制御可能な状態を維持する必要があり、そのために起こりそうなヒューマンエラーを理解する必要があります。  
オペレーターがある程度のスキルを身につけた、頻繁に実行されるタスクではエラーはまれであり、オペレーターがストレスを感じて驚いたときに起こりやすくなります。  
セキュリティシステムでも、重要であるがほとんど実行されないタスクで最も悪質な過失が予想されます。  
脅威と保​​護の一般的なメンタルモデルと一致する方法でリスクを提示する必要があり、ストレスに対するユーザーの反応の可能性は安全な結果につながるはずです。  
飛行機の設計者は、商業パイロットの免許を持つ誰からでも予測可能なスキルレベルに依存することができ、造船業者は商船の士官の長所と短所を知っています。  
プロフェッショナル側では、ユーザビリティテストをスタッフトレーニングと収益性の高い方法で統合できます。パイロットがシミュレーターでリフレッシュコースに行くと、インストラクターは、機器の故障、悪天候、客室の危機、航空管制の混乱など、あらゆる種類の組み合わせを投げます。それらで。  
このようなデータは、コックピット設計者への貴重なフィードバックです。  
それでも、ボーイング737Max飛行制御ソフトウェアなどの高額な災害があります。これには少なくとも1つの重大なバグがあっただけでなく、適切な障害モードと影響分析がありませんでした。時間–パイロットはあらゆる失敗に対処できると誤って想定されていました[89]。冗長システムの使いやすさをテストするときは、フォールトマスキングに注意を払う必要があります。出力が3つのプロセッサー間の過半数の投票で決定され、そのうちの1つが失敗した場合、システムは引き続き機能しますが、その安全マージンにはおそらく、オペレーターが正しく理解できない方法で侵食されています。  
したがって、障害がどのように残るかについて慎重に考える必要があります。DO178やDO254などの航空安全基準では、コモンモード障害を軽減するために、冗長性に加えて測定タイプ、物理、処理特性の多様性が一般に必要です。  
安全が重要なシステムからの別の教訓は、弁護士や規制当局の安全事例の一部として安全要件の仕様とテスト基準が必要になるが、両方を主流の製品ドキュメントと統合することは良い習慣であるということです。  
（これはボーイング737Max災害の要因でした。これは、飛行制御ソフトウェアの安全性の根底にあるユーザビリティの仮定が、737の以前のモデルから更新されなかったためです。）  
これについては次のセクションで説明します。  
2つの主な違いは、障害モデルにあります。  
人々は当然、敵の存在下でよりリスクを嫌います。これについてはセクション28.4で説明します。  
実際、私たちの仕事は、可能な限り最も不便な瞬間に微妙かつ悪意を持って間違った答えを出すコンピュータをプログラムすることです。  
これがセキュリティエンジニアリングが難しい理由の1つです。サタンのコンピュータはテストが困難です[1668]。  
27.4保護目標の優先順位付け  
注意深い安全分析または脅威モデリング演習は、これを知らせるためにいくつかの数値を提供することができます。  
また、環境またはコンテキストからのリスクの数値推定値を洗練するようにしてください。  
セキュリティの人々はしばしば前者に集中しすぎています。  
現在、利益は15％ではなく85％増の150万ドルです。  
多くの小売業者と詐欺管理戦略について話し合ったとき、最良の結果が得られた企業は、詐欺管理チームが財務ではなく営業に報告した企業であることに気付きました。  
したがって、詐欺エンジンを改善して3％しか拒否できなければ、売上は1％増加します。これは、最高マーケティング責任者の目を光らせる見込みです。  
同様に、オンラインサービスのサイト信頼性エンジニアは、システムの信頼性を高めすぎないようにしています。  
0.1％のエラーバジェットを意図的に設定する方がよいでしょう。これは、回復メカニズムを行使するために、時々故意の失敗を引き起こすなど、生産的に使用できます[236]。  
 ティックボックスアプローチは「もちろんオープンCVEがあってはなりません」と言うことですが、これはかなり高いコンプライアンスコストを課す可能性があります。  
CVEについては、後でセクション27.5.7.1で詳しく説明します。  
多くの場合、それはすでにきつすぎます。実際に行う必要があるのは、少しだけ焦点を合わせるだけです。  
20年前、多くのスーパーマーケットがセルフチェックアウトレーンを導入し始めました。  
アドバンテージを得た店舗の1つは、経験に照らして徐々に調整する、より寛容なアプローチから始まりました。  
2020年の初めまでに、開拓者はスウェーデンのLifvsのようなスタッフのいない小さなコンビニエンスストアです。アプリで店舗のドアを開け、購入内容をスキャンしてオンラインで支払います。  
私たちは、2020年のコロナウイルス封鎖の数か月に次の20年のイノベーションが押し寄せているのを見ました。 6月までに、他のスーパーマーケットから、スキャンアプリのダウンロード、購入品のスキャン、スキャン、カードへのチャージが求められています。  
リチャードシアーズが1880年代に「満足を保証するか、返金する」というスローガンを採用したとき、リチャードシアーズは破綻するだろうと誰もが思いましたが、現代の通信販売ビジネスを発明しました。  
しかし、成功した起業家は、セキュリティを迅速に改善する必要があるかもしれません。  
安全性のトレードオフはより困難です。  
しかし、安全行動によって明らかにされた私たちの実際の人間の生活の評価は、道路のジャンクションの改善のための約50,000ドルから、列車保護システムのための5億ドルを超えるまで変動します。これは、輸送政策の文脈においてのみです。  
この非合理性の理由はかなりよく理解されています。私はセクション3.2.5で心理学について、26.3.3で政策面について説明しました。この現象については、セクション28.4で説明します。  
27.5方法論  
1960年代までに、これはソフトウェアの危機として知られるようになりましたが、「危機」という言葉は、コンピュータの不安など、現在2世代続いた情勢には不適切かもしれません。  
開拓者たちは、基本的な科学の基礎とデザインルールのフレームワークを使用して、船や航空機を構築するのと同じ方法で問題を解決できることを期待していました[1420]。  
1960年代後半に戻ると、失敗した大規模なソフトウェアプロジェクトの数を、当時の30％程度から削減することを期待していました。  
現代のツールは、私たちが落ちる前に複雑さの山をさらに遠ざけますが、失敗の率はリスクに対する会社のマネージャーの欲求によって設定されます。  
ソフトウェアエンジニアリングは複雑さを管理することであり、そのうち2種類があります。  
そこで3これは、大ピラミッドの建設者にちなんで、「Cheopsの法則」として時々知られています。  
たとえば、銀行のコアシステムには、数千万行のコードが含まれ、複数の異なる配信チャネルを通じて販売された数百の異なる製品を実装する場合があり、1人では理解できません。  
最も重要なのは、マシン固有の詳細を扱う面倒な作業の多くを隠し、プログラマーが適切な抽象化レベルでコードを開発できるようにする高水準言語です。  
静的分析ツールなど、特にエラーが発生しやすい設計やプログラミングタスクをチェックできる正式な方法もあります。  
これらは、独自のツールセットによってサポートされます。  
  
システム開発の古典的なモデルは、1960年代にウィンロイスが米空軍向けに公式化したウォーターフォールモデルです[1628]。  
1970年代から2000年代半ばまでは、これが米国国防総省のすべてのシステムの開発方法であり、防衛だけでなく行政やヘルスケアを含む世界中の多くの政府がその主導権を握っていました。  
アイデアは、要件はユーザー言語で記述され、仕様は技術言語で記述され、ユニットテストは仕様に対してユニットをチェックし、システムテストは要件が満たされているかどうかをチェックします。  
 そして、次の2つで、それが適切に構築されているかどうか（検証）  
4つ以上のステップがある場合があります。一般的な詳細は、要件がさらに詳細な仕様に発展するにつれて、一連の詳細化ステップを持つことです。  
ウォーターフォールモデルの定義上の特徴は、開発が、要件の最初のステートメントから、フィールドへのシステムの展開まで、際限なく低下することです。  
 要件に対するシステムテスト。  
これは要件ですffi Validateffi仕様ffi Validateffi実装ffi＆ユニットテストffi Verifyffi統合＆ffiシステムテストffi Verifyffi Refineffi Codeffi Buildffi Fieldffi運用＆ffiメンテナンスffi図27.4：–ドイツのウォーターフォールモデルの標準であり、世界中の航空宇宙産業でも使用されています。自動車ソフトウェアの安全性に関するISO 26262規格に含まれています。  
それは別の動物よりも別の図です。  
ウィンロイスはまた、彼のモデルを、それが開発した規範的なシステムとしてではなく、混乱から秩序を取り始める手段として捉えました。  
それを機能させることができる場合、それは多くの場合、最良のアプローチです。  
時々、これは、コンパイラを書くとき、または（セキュリティの世界では）  
惑星間宇宙探査機の場合のように、1発しか得られないような外部的な理由から、トップダウンアプローチが必要になる場合があります。  
テクノロジーは変化しているかもしれません。環境が変化している可能性があります。または、プロジェクトの重要な部分は、おそらくいくつかのプロトタイプのテストを含む、人間とコンピュータのインターフェースの設計かもしれません。  
時々、正式なプロジェクトは遅すぎる。  
しかし、反復開発を使用する最も一般的な理由は、改善したい既存の製品から始めていることです。  
今日では、ソフトウェアが組み込みコード、アプリ、クラウドサービスの問題になっているため、すべてがさらに複雑になっているため、多くの企業では「メンテナンスは製品」という現実があります。これでウォーターフォールモデルが古くなることはありません。それどころか、プロジェクトを管理して主要な新機能を開発したり、既存のコードをリファクタリングしたりするためによく使用されます。  
  
反復的な開発には、新しい製品の仕様を確定するための迅速なプロトタイピングの練習から、既存のシステムを修正または拡張するための管理されたプロセスに至るまで、さまざまな風味があります。  
バリーベームによって考案されたこのプロセスは、図27.5のように描かれることが多いため、スパイラルモデルと呼ばれています。  
セキュリティエンジニア向けのプロトタイプアプリケーションは、セキュリティユーザビリティテストベッドから概念実証攻撃コードまで多岐にわたります。  
ここで、アジャイル開発として説明する2番目のケースは、合計される場合があります。あなたの最悪の問題。  
進化的アプローチの初期の提唱者はHarlan Millsでした。彼は、機能する最小のシステムを構築し、実際のユーザーで試してから、少しずつ機能を追加する必要があることを教えました。  
 ゼロから。  
進化的開発に関する画期的な初期の本は、1994年のMicrosoftのSteve Maguireによる「Debugging the Development Process」でした[1209]。  
マイクロソフトは、そのアプローチをIBMのアプローチと対比させました。 IBMエコシステムでは、ウォーターフォールアプローチが主流でした。  
 それ以来、四半世紀に専門家としての実践が発展し、進化的発展は現在「アジャイル」として知られていますが、それは明らかに同じ獣です。  
コアテクノロジーは回帰テストです。  
回帰テストは、以前は機能していたものがまだ機能するかどうか、古いバグが戻っていないかどうかをチェックします。  
アプリがクラウドサービスと通信する必要がある場合や、車両内の複数の電子コンポーネントが連携する必要がある場合、または単一の車両コンポーネントをカスタマイズする必要がある場合など、システムが連携する必要がある場合、状況は少し複雑になります。いくつかの異なる車両で動作します。  
しかし、いずれにしても、ベータテスト用に出荷できる実行可能なコード、またはプロセスの次の段階が何であれ、常に実行可能なコードがあります。  
自動化された回帰テストが広く使用される前は、IBMのエンジニアは、バグ修正の15％が新しいバグを導入したり、古いバグを再導入したと考えていました[18]。  
たとえば、IBMはシステムアナリスト、プログラマー、テスターの役割を分けていました。アナリストが顧客に話しかけ、プログラマーがコーディングした設計を作成し、次にテスターがコードのバグを探しました。  
これは遅く、コードが肥大化しました。  
これは、多くのバグを書いた悪質なプログラマーを抑え、より巧妙で注意深い開発者によってより多くのコードが作成されるようにしました。  
  
2000年代初頭までに、Microsoftは主要なテクノロジー企業としてIBMを追い抜いていましたが、WindowsとOfficeのセキュリティの脆弱性に対する批判が高まり、マルウェアがますます増えました。  
最終的に2002年1月に、ビルゲイツはすべてのスタッフに「信頼できるコンピューティング」のメモを送り、機能よりもセキュリティを優先し、エンジニアがセキュリティトレーニングを受けている間はすべての開発を中止するように指示しました。  
脅威のモデリングについては、セクション27.3.5ですでに説明しました。安全な開発に対する彼らの最初の取り組みは、2002年にMichael HowardとDavid LeBlancの「Writing Secure Code」[927]に登場しました。  
 2008年に登場し、Windows開発者に広く採用されています。  
「ソフトウェアの脆弱性の数と重大度を減らすことを目的とし」、「開発プロセスのすべてのフェーズ全体でセキュリティとプライバシーを導入します」。  
次に、5つのSDLコンポーネントがあります。  
要件：これには、リスク評価と、特定の種類のフローが含まれている場合にコードが次の段階に進むのを防ぐ品質ゲートまたは「バグバー」の確立が含まれます。  
2。  
3。  
4。  
5。  
すべての開発者に基本的なセキュリティトレーニングを提供するだけでなく、組織的な側面もいくつかあります。開発チームの外から、そしてチーム内のセキュリティまたはプライバシーの擁護者から、すべてが完了したことを確認します。  
1989年以降、ワッツハンフリーは能力成熟度モデル（CMM）を開発しました  
能力は、個々の開発者ではなくチームの機能であるという考えに基づいています。  
開発者は、さまざまなコーディングスタイル、コードのコメントと書式設定のさまざまな規則、APIのさまざまな管理方法、さらにはさまざまなワークフローリズムから始めます。  
これには、個々のエンジニアに継続性を提供し、チームの総合的な専門知識を維持するために、リソース割り当ての効率をある程度犠牲にする自己規律が必要です。  
  
これは、進化的開発を推進している最大の企業がセキュリティのウォーターフォールアプローチに戻ったことを示しています。  
セキュリティのプロパティは創発的かつ多様であり、セキュリティエンジニアの数は少なく、ツールへの投資もそれほど多くありません。  
ただし、回帰テストは十分ではありませんが、変更によって影響を受ける機能を見つけるため、必要です。  
このため、Windowsへのセキュリティパッチはゲーテッド開発の例です。定期的に、製品のプレリリースバージョンが一連の追加のテストとレビューの全体にプッシュされ、リリースの準備がされています。  
準備には、Windowsの場合はさまざまな周辺機器やアプリケーションを使用したテスト、規制対象製品のソフトウェアの場合は再認証が含まれる場合があります。  
それらは、変化する環境、進化する脅威、新旧のプラットフォームへの新しい依存関係、およびその他のものの束によって駆動されます。  
ここでもMicrosoftが先駆者でした。  
したがって、単一のバグを修正するだけでなく、ツールチェーンを更新して、製品全体で同様のバグをすべて見つけて排除します。  
それから2015年まで、企業から自宅のPCやタブレットのユーザーまで、すべてのお客様のソフトウェアが毎月第2火曜日に更新されました。  
最新の品質のツールは、CVEが開いているコードがないことを確認するのに役立ちます。そのため、このようなツールを使用しているすべての顧客もパッチを適用する必要があります。  
2015年以降、Windowsホームユーザーは継続的な更新を受け取ります4。  
安全性は、セキュリティと同様に、システム全体の創発的な特性であり、構成するものではありません。  
しかし、接続されたデバイスがセキュリティなしに安全性を持つことは困難であり、車などのデバイスがインターネットに接続されている今、それらもパッチサイクルを取得しています。  
たとえば、自動車には数十の電子制御ユニット（ECU）が含まれる場合があります。  
自動車業界の企業は相互に不審であり、NDAのもとでもソースコードを互いに共有しないため、テストは複雑になる可能性があります。  
これらは実際の費用がかかり、実際の車両を4台維持する必要もあります。これも事態を打破します。かつて、私たちが使用していたWindowsラップトップがそれ自体を更新したときに、TVのクルーにボディモーションキャプチャスーツを使用した実験をデモしようとしていました。突然、キャプチャソフトウェアが機能しなくなります。  
ロードテスト。  
これは、テスラが大きな利点を持っている点の1つです。 Teslaは、ソフトウェアをビジネスの中核とする技術企業として、ソフトウェア開発の大部分をコンポーネントサプライヤーに任せているため、変更を数週間でテストして出荷することができます。  
これの持続可能性の側面については、次の章で説明します。  
  
2010年代初頭以降、中央サーバーでホストされ、ユーザーに販売して配布するのではなく、シンクライアントからアクセスし、サブスクリプションベースで支払うソフトウェアが増えています。  
ベンダーがすべての顧客からこれらのタスクを引き継ぐことができれば、重複するコストの多くが取り除かれ、専門知識を備えているため、物事をよりよく管理できます。  
Software as a Service（SaaS）の背後にある主要な技術革新  
何千もの顧客が数十の異なるバージョンのソフトウェアを管理するのではなく、ベンダーは少数の顧客を新しいバージョンに移行してテストし、残りを移行することができます。  
一部の企業は現在、1日数回展開しています。これは、小さな変更を頻繁に行う方が、Microsoftのパッチ火曜日などの大規模な展開よりも安全で、何かを壊すリスクが少ないという経験があるためです。SaaS会社は通常、ロードバランサーの背後にあるVMで実行されている多数のサービスインスタンスでソフトウェアを実行します。これにより、実行中のサービスを管理するための間接的なポイントが提供されます。  
ローリングデプロイメントを行うために、ロードバランサがトラフィックの約1％を新しいバージョンのインスタンスに送信するように構成します。これは、鉱山労働者が一酸化炭素の漏れを検出するために使用されたケージの鳥の後に「カナリア」と呼ばれることが多いものです。  
ロギングシステムが問題を検出すると、開発者に警告が出されます。  
後方互換性を損なう変更を行う場合、通常は新旧両方のシステムで機能する中間段階を構築します（とにかく1980年代に銀行のメインフレームの世界でこれを行っていました）  
段階的なリリースとローリング展開を通じてリスクを管理する機能は、テストの経済性を変えます。  
また、ユーザーが行うすべての操作を確認できるため、セキュリティ、安全性、および収益の観点から、ユーザビリティがどのように失敗するかを初めて理解することができます。  
分析コレクターは、すべての行動イベントをログに書き込みます。ログは、メトリック、分析、クエリのデータパイプラインに送られます。  
広告主導のサービスは、アクティブユーザー、ユーザーセッションあたりの時間、特定の機能の使用などのエンゲージメント指標によって最適化できます。  
そのような改善は通常、それ自体ではかなり小さいため、それらを測定するには制御された実験が本当に必要です。しかし、あなたがそれらの多くを行うとき、それらは合算します。  
そのため、大企業は、小規模の競合他社よりも迅速に製品を最適化できます。 SaaSは、他の多くのデジタルテクノロジーと同様に、短期的にコストを削減するだけでなく、長期的にロックインを高めます。  
複数の同時実験を実行する一方で、システムの機能拡張を展開することについては、多くの面倒な詳細があります。  
これにより、クラウドネイティブ開発またはDevOpsとも呼ばれるコードとしてインフラストラクチャの世界に移動します。すべてがコンテナやVMなどで開発されるため、すべてのインフラストラクチャはコードに基づいており、すばやく複製できます。  
新しいコードをテストインフラストラクチャに迅速に展開し、現実的にテストできます。  
ソリューションは、アプリケーション開発プロセスの一部として、クラウドプラットフォームAPIを使用してアプリケーション自体と関連するインフラストラクチャをデプロイし、監視メカニズムにフックできるようにするデプロイメントコードを記述することです。  
私が知っているこれに対する最良のガイドは、Googleの2013年の本「サイト信頼性工学」です。 SREは、DevOpsの用語です[236]。  
2000年代から2010年代初頭にかけて、彼らは他の誰よりも大規模に運営されていたため、より多くのタスクを自動化する必要があり、上手になりました。  
中心的な戦略は、ソフトウェアエンジニアリング手法を適用してシステム管理タスクを自動化し、迅速なイノベーションと可用性を両立させることです。  
現実的なエラーバジェットを設定した場合、たとえば0.1％または0.01％の可用性がない場合、それを使用して多くのことを達成できます。  
残りのエラーバジェットを使用して、実験的なフレームワークをサポートし、制御された停止を実行して依存関係をフラッシュします。  
 セクション12.2.6.2で、技術的負債に言及しました。  
ドキュメントを読み飛ばしたり、汚くて汚い問題を修正したり、修正を徹底的にテストしたり、セキュリティコントロールの組み込みに失敗したり、エラーの結果を処理できなかったりするたびに、問題を保存します。将来的に利息を返済する必要があるかもしれない[41]。  
時間の経過とともに、システムは借金に深く陥り、保守や使用が困難になります。それらはリファクタリングまたは交換する必要があります。  
したがって、技術的負債の管理は本当に重要です。これは、この本の第2版以降のシステム管理の考え方の変化の1つです。  
  
私が書いているように、2020年の最先端は、アジャイルアイデアと方法論を開発と運用だけでなく、セキュリティにも適用しています。  
 しかし、新しい脅威、環境の変化、意外な脆弱性への対応。  
たとえば、セクション12.2.2で、DevOpsは、銀行が何年にもわたって依存してきた開発と生産の分離を弱体化させることを述べました。職務の分離が必要な場合は、それを再考する必要があります。以下では、Microsoftの世界とGoogleの世界と大まかに呼ぶ2つの例を示します。  
  
銀行や保険会社から小売、出荷、鉱業に至るまでの世界最大の商業企業のほとんどは、過去25年間にWindowsでエンタープライズシステムを構築しており、現在システムインテグレーションやファシリティ管理会社を使用して実際の作業を行っているAzureに移行しています。 。  
ここでのポリシーは主に、内部統制機能の標準セットに加えて、安全な開発ライフサイクルを要求するためにMicrosoftをフォローするBig Four監査人によって設定されています。  
ここでのDevSecOpsタスクは、これらの管理タスクを自動化することにより、ツールを段階的に統合することです。  
現在、同社は、エンジニアリングへの投資の10％がサイバーセキュリティへの投資であると主張しています。  
セキュリティチームが引き続きゲームを開発し、範囲を拡大し、有効性を高めるには、適切な自動化が不可欠です。それがなければ、彼らはどんどん後ろに落ちて、燃え尽きます[1846]。  
開発戦略の1つは、開発プロセスの早い段階でセキュリティの専門家を関与させ、後で遅延を回避することを含めて、物事を「すぐに失敗させる」ことです。不正な認証、暗号機能の使用ミス、注入の機会などのエラーを検出するための専門ツールを購入または構築する。新しいバージョンの自動および手動のセキュリティテスト。また、ステージングネットワークのスキャンや、資格情報、暗号化キーなどのチェックを含む、構成と展開の自動テスト。  
これらについては、後で詳しく説明します。  
27.5.6.2 Googleエコシステム  
DevSecOps戦略は、Amazonでも多少似ていますが、自社の製品向けに最適化されています。 [1966]のCTO Werner Vogelsによって説明されています。  
このセクションでは、彼らの本と、最近大手サービス会社で働いている同僚を利用しています。  
そのため、施設管理会社はサポート機能を統合してコストを節約し、エラーを減らすことに取り組んでいますが、主要なサービス会社は信頼性を重視しています。  
これにより、回復可能性の設計、理解可能性の設計、人間が可能な限り本番システムに触れないようにするという願望などのさらなる原則が推進されます。  
このようなリスクは、適切なツールプロキシを構築して管理します。  
安全なテストを行うと、さらに複雑になります。  
 大量の個人情報を含むシステムをどのようにテストしますか？  
 これらのほとんどは、1980年代のメインフレームの世界ですでに対処しなければならなかった質問ですが、それらはたまにしか発生せず、人間の創意工夫といくつかの主要なスタッフを信頼することによって対処されました。  
まだ緊張があります。  
アプリケーションレベルでは、システムは、防御可能なセキュリティ境界と改ざん防止セキュリティコンテキストを備えたマイクロサービスコンポーネントにますます区分化されているため、アリスがショッピングシステムのカタログを危険にさらした場合でも、支払いサービスは独立しているため、ボブとしてお金を使うことはできません。 。  
このようなドメインを使用すると、あらゆる妥協の爆発範囲を制限できます。理想的には、システム全体を稼働させることなく侵入に対処できるようにしたいと考えています。  
失敗したドメインが失敗したとき、新しいものを起動するのはいつですか、いつ何か違うことをしますか？  
 どのコンポーネントがフェールオープンで失敗し、どのコンポーネントがセキュアで失敗する必要がありますか  
 負荷制限とスロットルの役割は何ですか？  
 一部の広告を破棄しますか、ログオンに追加のキャプチャが必要ですか、またはその両方ですか？  
 大企業は、そのようなサービスのアプリケーションフレームワークの構築に多くのエンジニアリング時間を費やしています。  
すべてのhttpを終了する単一のフロントエンド  
単一のフロントエンドは、ロードバランシングとDDoS保護、および多数の異なる言語のサポートなどの他の多くの機能のための単一の場所を提供することもできます。  
理解しやすい構造によるセキュアAPIがある場合は、それが最適です。  
Googleクラウド、AWS、Androidキーストアのいずれの場合でも、キー管理サービスを使用する必要があります。  
  
このようなプロセスを調整するためにMicrosoftのアプローチ、Googleのアプローチ、または独自のアプローチのいずれを使用するかに関係なく、メトリックが必要です。適切な候補には、開発チームに開かれたセキュリティチケットの数、セキュリティが失敗したビルドの数、およびかかる時間が含まれます。関連する規制（SOX、GDPR、HIPAAのいずれか）に基づくコンプライアンスを達成するための新しいアプリケーションの場合Dev、Sec、Opsが収束すると、メトリックと管理プロセスは、セクション21.4で説明したネットワークモニタリングメカニズムからセキュリティインシデントおよびイベント管理まで、ネットワークフェンスメカニズムに収束します。  
よく運営されている企業は、プライバシーへの影響の評価、アクセス制御の改善、ロギングの拡張などを行うために行うスプリントを介して、すべての開発/運用スタッフにセキュリティプロセスをより可視化できます。  
競合が組織の管理構造の機能として自然に発生する場所を理解し、何らかの形でそれらを建設的に保つことが重要です。  
バグがエクスプロイトになってから攻撃になり、これらの攻撃が発見されて脆弱性レポート、ファイアウォールなどのデバイスを使用した暫定的な防御、そしてユーザーだけでなく複雑なサプライチェーンに沿って展開される明確なパッチが発生するプロセスセキュリティ管理の中核です。  
27.5.7脆弱性サイクル  
攻撃対象を制限するアーキテクチャと正式な方法の適用を組み合わせることで、脱出できると期待されていました。  
2000年代初頭までに、パッチサイクルをより適切に管理する必要があるという結論に達し、セキュリティ侵害の開示に関する法律、CERT、および責任ある開示の最新のアプローチは、この期間に組み込まれました。  
研究者は、顧客、学者、国家情報機関の請負業者、または犯罪者でさえあります。  
脆弱性市場の考え方は、2000年にJean CampとCatherine Wolframによって最初に提案されました[371]。企業は脆弱性を購入するために設立され、やがていくつかの市場が出現しました。  
 法医学会社（法執行機関に販売している）  
そのような事業者は現在、AndroidとiOSの永続的なリモートエクスプロイトに数百万ドルを提供しています。  
ポピュラーなプラットフォームに対するゼロデイエクスプロイトの市場価格は6桁から7桁に上昇しているため、バグの報奨金も発生しています。  
2019年には、バグ報奨金プラットフォームHackerOne [2030]だけで、少なくとも6人のハッカーが100万ドル以上稼いでいることが明らかになりました。  
このようなサプライチェーン攻撃は、以前は国民国家の保護でした。現在、彼らは開放しています[890]。  
それが十分に使用されれば、最終的に誰かが気付くでしょう。攻撃が報告された後、ベンダーがパッチを発行します。パッチはリバースエンジニアリングされて、他の多くのアクターがエクスプロイトコードを利用できるようになります。  
さまざまな利害関係者の関心が根本的に分岐する可能性があるため、パッチ適用サイクルを正しくすることは、他の何よりも情報セキュリティの経済学における問題です。  
ベンダーは、パッチの費用を節約するために、バグがまったく見つからないことを望んでいました。  
確かに、パッチが必要な顧客のデバイス（車など）でコードが使用されている場合  
2。  
怠惰な顧客はパッチの適用に失敗し、結果として感染する可能性があります。  
 3。  
4。  
5。  
6。  
優れたものは、Microsoftのパッチ火曜日などの定期的なイベントに対処するための自動化を構築していますが、オフィスや工場の膨大な数のIoTデバイスの更新やリスク評価は、今後数年間の頭痛の種になるでしょう。  
1990年代の議論は、製品のパッチを数か月または数年も放置しておかれたというソフトウェアベンダーに不満を抱いていた人々によって動かされました。  
これにより、研究者がベンダーに提供する必要のある休息スペースについて、さまざまな提案がなされ、「責任ある開示」に関する議論が生まれました[1572]。  
5とCERTのグローバルネットワークは、脆弱性が公開される前にパッチが発行されるのを遅らせて、ベンダーに通知します。  
ああ、そして機関は地元のCERTにホットラインを持っていたので、自然に発生するエクスプロイトを事前に知っており、それらを悪用できる可能性があります。  
  
産業的側面はCommon Vulnerabilities and Exposures（CVE）です  
これはMitreによって維持されますが、大規模ベンダーへのCVEの割り当てを委任します。  
Common Vulnerability Scoring System（CVSS）があります  
これを計算する方法は、時間とともに着実に複雑になり、攻撃がローカルアクセスを必要とするかどうか、その複雑さ、必要な労力、影響、エクスプロイトコードとパッチの可用性、ターゲットの数と潜在的なダメージ。NISTのNational Vulnerability Database（NVD）  
政府の脆弱性リソースおよび業界リソースへの参照を提供する」は、CVEリストに基づいています。  
現在、非常に多くの脆弱性が報告されており、何百ものパッチが出荷されているため、自動化が不可欠です。  
伝統主義者は、バグは多く、相関がないため、ほとんどのエクスプロイトは既存のパッチからリバースエンジニアリングされた脆弱性を使用しているため、最小限の開示しかないと主張しました。  
この議論については、セクション8.6.2で説明しました。  
なんらかの調整を経て、物事を行う現在のコンピューター業界の方法は10年以上安定しています。  
27.5.7.2調整された開示  
セクション4.3.1で、フォルクスワーゲンがバーミンガム大学とナイメーヘン大学の学者を発見し、責任を持って開示した後に、オンラインで利用できる自動車盗難ツールですでに悪用されていたフォルクスワーゲンのリモートキーエントリシステムの脆弱性をどのように訴えたかについて説明しました。  
マイクロソフトやグーグルなどの企業は、バグ報奨金プログラムや毎月のパッチ適用が人々を訴えると脅すよりも効果的であることを学ぶのに20年を費やしましたが、レガシー業界の多くの企業は、製品にさらに多くのものが含まれているにもかかわらず、まだこれを解決していませんその他のソフトウェア。  
このようなサプライチェーンの問題の結果として、責任ある開示は調整された開示に取って代わられました。  
音声認識とジェスチャー認識を行う場合は、LinuxまたはFreeBSDのいくつかの動作を実行するアームチップが含まれ、別のLinuxの動作を実行するクラウドサービスと通信し、AndroidまたはiOSで実行されるアプリによって制御できます。  
その対応は、EU全体での即時の安全想起でした[654]。  
何十もの組み込み製品で使用されているプラ​​ットフォームで、悪用可能なバグを誰かが発見した場合はどうなりますか？  
。  
しかし、目覚まし時計からTVまで、そして子供のおもちゃから地雷まで、Linuxを組み込んだ製品は何千もあります。  
プラットフォームでの開示の調整は、深刻な問題の1つです。  
そのようなショックへの対処は、2010年代後半に独自の専門分野となったプロセスの1つの側面にすぎません。つまり、セキュリティインシデントとイベント管理です。  
27.5.7.3セキュリティインシデントおよびイベント管理  
昔は、ベンダーは製品の新しいバージョンで応答するのに数か月かかることがあり、警告（または拒否）を発行する以外に何もしないことがよくありました。  
現在、米国とヨーロッパの両方の違反通知法により、企業は個人のプライバシーが侵害された可能性のある攻撃を開示することを義務付けられており、人々は問題が迅速に修正されることを期待しています。  
最初に、できる限り早く脆弱性について学習するようにしてください。できれば悪者（または報道機関）よりも遅くないことが望ましいです。  
これは、脅威インテリジェンスチームの構築を意味します。  
顧客の意見を聞くことは重要です。顧客がバグを報告するための効率的な方法が必要です。  
セクション27.5.7で説明されているバグバウンティ、脆弱性市場、CERT、CVEのより大きな技術的エコシステムに関与する必要があります。  
20年前は、午前3時に何かを修正する必要がある場合に備えて、各製品チームの1人のメンバーがポケットベルで「オンコール」になることを意味していました。  
これは、セクション21.4.2.3で説明した侵入検知機能とネットワーク監視機能から脅威インテリジェンスチームまで、責任を負う開発チームを特定し、上流のサプライヤと下流の顧客の両方に通知するまで拡張されます。  
衛星電話が必要かどうか考えるのをやめたことがありますか？  
顧客が1日に1回Webサイトにアクセスしてアップグレードを確認するのは簡単に思えるかもしれませんが、自社のシステムがデバイスに依存していて依存関係をテストする必要がある場合、緊張があります[195]：パッチをすばやく適用する先駆者は、システムを壊す問題を発見する一方で、テストに時間をかける人は攻撃に対してより脆弱になります。  
運用は非常に重要です。テストされていない緊急パッチプロセスは、害を及ぼす可能性があり、緊急時には通常のパッチプロセスをできるだけ速く実行することが経験からわかっています[23]。CEOに連絡してすぐに説明するメカニズムが必要です。そうすれば、CEOが管理下にあることを示し、主要な顧客を安心させることができます。  
そして、あなたはマスコミに対処する計画が必要です。  
重大度の異なるインシデントに備えて一連のプレスリリースを用意し、CEOが適切なものを選択して詳細を入力するだけでよいようにします。  
 ジャーナリストが呼び出します。  
訴えられることを期待できます。  
（それはよく起こるかもしれません。300万のアカウントのうち、とにかく数万のアカウントが毎年何らかの詐欺に見舞われ、銀行がそのすべてに対してあなたを訴えます。）  
それが複数回発生した場合、顧客の喪失が予想されます。顧客のチャーンは、1回の通知違反後は2％に過ぎませんが、2回後は30％になり、3回後にはさらに高くなります[2037]。  
  
組織の問題は、組織のメモリが失われたり、変化する環境を監視するためのメカニズムが不足したりする場合のように、システム障害の単なる要因ではありません。  
セクション8.6.7で触れた、組織での人々の行動に関する大規模な文献があり、さまざまな章でさらに多くの例を示しました。  
ベゾスの法律では、2人のピザから供給されるよりも多くの人々と一緒に開発プロジェクトを実行することはできません。  
プロジェクトに複数のチームが含まれる場合、メンバーはランダムに互いに話すことができないか、混乱を招きます。また、帯域幅がないため、すべての通信を最下位の共通マネージャーを通じてルーティングすることはできません。  
横方向の相互作用がないN人の場合、クリーンな軍事指揮系統の通信の複雑さはlog Nです。誰もが他の誰にも相談しなければならない場合、それはN 2です。サブセットが問題について考える委員会を形成できる場合、2Nに向かう可能性があります。  
多くの大規模な開発プロジェクトがクラッシュして焼失しました。  
大規模なソフトウェアプロジェクトの災害に関する古典的な研究は、Bill Curtis、Herb Krasner、およびNeil Iscoe [504]によって書かれました。  
学部の講義で私が挙げる例は、ロンドンの救急車サービスの新しい派遣システムのメルトダウンです。そこでは、過度に野心的なプロジェクト、不十分な仕様、そして実際のテストが行​​われなかったため、1日の救急車カバーがないままになりました。  
その日はたまたまロンドンにいたので、覚えています。  
（実際、プロジェクトの失敗に関する多くのケーススタディを読むことを強くお勧めします。）  
多くの大規模な商業および政府システムが、2000年に備えて2桁の日付を4桁の日付に変更するための大規模な修理作業が必要であり、大規模な開発プロジェクトのかなりの割合が遅れるか、または決して行われないという従来の経験を受け入れる場合結局のところ、多くの人々は、かなりの数のシステムが1999年の終わりに故障すると予想し、広範囲にわたる混乱を予測しました。  
確かに、中小企業が使用するシステムに対するリスクは誇張されていました。私たちは大学ですべてのシステムを徹底的にチェックしましたが、かなり簡単に修正できないものは何も見つかりませんでした[69]。  
しかし、重大な失敗の報告はありませんでした。  
Y2Kバグ修正の要件は完全にわかっていました。「このシステムを、現在と同じように、2000年以降も引き続き機能させたい」。  
セキュリティの達人である必要はありませんが、何を構築しようとしているか、どのように機能するかを理解する必要があります。」組織は、目標を設定し、それを満たす計画を立てる際に邪魔になるため、不確実性に対処することが困難です。  
プロジェクトの不確実性を管理する方法に関する重要なビジネススクールの文献があります[1179]。  
難しい問題に直面すると、人々は関連しているが簡単な問題を猛烈に攻撃するのが一般的です。セクション26.2.8など、いくつかの例を見てきました。  
次のシットストームがどこから来るのか、本当にわかりません。  
2010年代半ばに、競合企業がAmazonデータセンターの同じマシンでワークロードを実行するのに十分なCPUが確保されていると考えました。それからスペクターがやってきた。  
ムーアの法則は鈍化していますが、さらに多くの驚きがあります。  
しかし、チェックリストは管理者の注意と労力をあまり必要とせず、質の高い官僚はそれらを愛しています。多くの場合、結果はおかしいです。  
また、人々はチェックリストをゲームする方法を急速に学びます。  
脅威モデルが政治的に敏感な場合、事態はより複雑になります。  
インサイダーは、その一部が悪意のあるものであったり、ほとんどのものが不注意であったりするため、多くの場合、最大のセキュリティリスクです。  
あなたはコントロールのためのスイートスポットを見つける必要があり、それはしばしばそれを文化に組み込む方法を考え出すことを意味します。  
組織内でリスクエコシステムを正しく設定するには、微妙な点と粘り強さの両方が必要です。  
 追放すること（どの半導体企業が防ぐのに苦労している）  
現時点でのエグゼクティブサマリーは、ビッグデータ侵害の被害を受けたほとんどすべての企業がISO 27001認証を取得していることを示していますが、監査人が何かが大丈夫ではないと言ったために失敗しました。  
。  
 セクション12.2.6.3で説明したように、トランザクションの内部統制構造は監査人の影響を大きく受ける可能性がありますが、より広範なセキュリティ文化はさまざまであり、問​​題です。  
一流のテック企業とフィンテック企業にはいくつかのスターがいますが、CISOであることは感謝の仕事です。  
多くの組織では、昇進は年功序列と連絡先の問題です。したがって、CEOになりたい場合は、途中であまり多くの人を怒らせることなく、20年かけて階層を登らなければなりません。  
この仕事はまた、多くのストレスと燃え尽き症候群のリスクをもたらします。 CISOの平均在職期間は約2年です[430]。  
彼らがそれを重要だと思わなければ、CISOにはチャンスがありません。  
リスクエコシステムを歪めることができる1つの方法は、企業が管理しているシステムのリスクの一部がサードパーティに捨てられるようにすることです。  
これについてはセクション12.5.2で、銀行が決済システムの不正責任をカード所有者、販売者、またはその両方にシフトしようとする状況で説明しました。  
もう1つの例はヘンリーフォードで、彼の車の1つでけがをした場合は、彼ではなくドライバーを訴えるべきだと考えていました。裁判所や国会議員が製造物責任を明確にするのに数十年かかりました。  
重要な幹部の個性は重要です。  
問題のもう1つの原因は、システム設計の決定が、その責任を負う可能性が低い人々によって行われる場合です。  
ITスタッフの離職率は高く、契約スタッフに大きく依存しています。冗長性に対する恐れは、忠実なスタッフを不正な求職者に変える可能性があります。  
したがって、大規模なシステムプロジェクトに取り組むときは、周りを見回して、後で問題が発生したときに誰が責任を取るのかを自問自答してください。  
実際、企業が完全に安全な製品を入手すると決定した場合、彼らは複数の専門家を雇うべきです。  
もちろん、これはめったに起こりません。  
27.6チームの管理  
現代のシステムの多くはすでに非常に複雑であるため、システムのすべての側面に対応できる開発者はほとんどいません。  
 これは50年以上にわたって激しい議論の的となっており、さまざまな作家が個人的なスタイルや企業文化を反映しています。  
  
1960年代にさかのぼって、フレッドブルックスの有名な著書「The Mythical Man-Month」は、世界初の大規模なソフトウェア製品であるIBM S / 360メインフレームのオペレーティングシステム[328]の開発から学んだ教訓を説明しています。  
一部のプログラマーは他のプログラマーよりもはるかに生産的であると考えていたため、管理職に昇進させて「失う」のではなく、上級管理職の給与と尊敬を込めて投稿を作成しました。  
Microsoft、Google、Facebookなどのより現代的な企業が取った見解では、最初は超生産性の高いエンジニアのみを採用したいと考えています。特に、年間100万人のCVを獲得し、20,000人の新しいエンジニアのみを採用する予定の場合はそうです。  
生産性も文化の問題です。ある会社で成功するエンジニアは、別の会社ではあまりうまくいかない場合があります。  
エリート大学はハイテク専攻と同じような状況にあり、各場所に何十人もの応募者がいます。私たちは長年にわたって、採用と入学の多様性を監視するメカニズムを持つことを学びました。  
現代のテック企業は、国際的に知られているデザイナーから、チューリング賞を受賞したコンピューターサイエンティストまで、複数のテックスーパースターを雇っています。  
人生をソフトウェアの作成に費やすことを望んでいて、ソフトウェアを上手に利用したい人は敬意を払う必要がありますが、給与、ボーナス、株式、またはエグゼクティブダイニングルームへのアクセスなどのフリッパーなど、組織はそれを示しています。ハイテク企業もそれを手に入れ、1つまたは2つの銀行が始めました。  
英国の公務員のモットーは、「科学者は上手くいってはならないが、一番上にいてはならない」です。そして、私が知っている複数の自動車会社が、まともなソフトウェアエンジニアの雇用と維持に大きな問題を抱えています。  
。  
  
コンピューティングの初めには、多くの女性プログラマーがいました–彼らは1960年代後半まで大多数であり、Grace HopperやDame Stephanie Shirley（「Steve Shirley」として何年も会社を経営していた）などの開拓者が含まれていました  
私が1970年代の初めに始めたとき、今日よりもずっと良い性別バランスがまだありました。  
しかし、物事はアメリカとイギリスで男性優位になっています。  
しかし、以前は東ヨーロッパだった共産主義国では、その比率は約3分の1です。  
 インドでは、ジェンダーのバランスが近いです。  
それはロールモデルの欠如ですか、それとも学校のキャリアアドバイザーのせいですか、それとも多くのITショップは女性にとって不愉快な職場環境ですか  
コンピュータサイエンスの分野でも、多くのサブカルチュラルバリエーションが見られます。  
。  
ロールモデルと歴史は重要です。  
より多様なチームがより効果的であり、真の変化は、最初に採用した女性ではなく、チームカルチャーを変えるだけの十分な時間がある場合に生じます。  
また、より賢明なマネージャーを獲得することも意味します。  
もっと微妙に、もっと多くの女性を引き付けたい場合は、1960年代に故カレン・スパールク・ジョーンズが始めた自然言語処理グループで、ジェンダーのバランスを取っています。  
スタッフを保護し、スタッフが得意なことを行うためのスペースを確保する必要があります。  
このような人々は、技術的に何が起こっているのかを理解していないことがよくあり、誰が生産的であるかわからず、計時や自分自身をどれだけ統合するかによって人々を判断しなければなりません。  
  
現代の開発には、チームをまとめて維持したいという欲求と、チームをより効率的かつ予測可能なものにすること、そして人々を動かしてスキルを開発し、古くならないようにすること、そして重要なすべてを維持できる一人以上の人がいることとの間に緊張があります。  
たとえば、アプリを作成している場合、数人がAndroidコードを、2人がAppleコードを、そして2人がサーバーを作成することもできます。  
継続的インテグレーションを行う場合は、A / Bテストを専門とするエンジニアがいますが、ゲーテッドアプローチを使用している場合は、サードパーティ製品との互換性やセキュリティ認証に重点を置く可能性があります。  
また、2人のピザからフィードするよりも多くの人でプロジェクトを実行することはできないため、一部の人にこれらのスキルを2つ以上持たせたいと考えています。  
 [1209]。  
ここでも、労働慣行は長年にわたって進化してきました。  
彼らは彼らがやったこと、彼らがやろうとしていること、そして何が問題なのかを説明しています。  
私たちのチームでは、毎日のランチと、週に1回の正式な進捗会議を組み合わせました。  
 バグを発見した人（自分でコーディングしたバグも含む）がいる場合、それは悪い習慣です。  
バグ追跡が重要であり、適切な統計を保持できるチケットシステムは品質を向上させるための重要なツールです。  
ソフトウェアはそれほど劇的ではありませんが、違いはありません。開発者に、エラーを含め、経験を共有してもらいやすくする必要があります。  
管理が不十分なチームの1つの兆候は、コードベースが混沌としたスタイルの混合であり、誰もが独自のことをしていることです。  
無駄な時間は別として、再フォーマットされたコードは分析ツールを作動させる可能性があります。  
あなたは脆弱性を書いたプログラマーが彼らが何をしていたと思ったのか知りたいです：それは設計エラーですか、それともコーディングの失敗ですか？  
したがって、プロジェクトを開始するときは、全員に座って、午後にあなたの家のスタイルがどうなるかを考えさせてください。  
このスタイルを作成することは、午後のペイントボールを費やすことや、最新の企業のチームビルディングの流行を起こすことよりも優れたチームビルディングアクティビティです。  
27.6.4新しいプロパティ  
アビオニクスなどの分野の安全性にも同じ質問が当てはまります。  
前の世紀にIBMが行ったように、これらの機能をアナリスト、プログラマー、およびテスター間で分割するよりも、開発者が独自の設計を進化させ、独自のバグを修正する方が責任があるとMicrosoftがより効果的であるとマイクロソフトが認識したことはすでに述べました。これらは、修士号または同等の内部資格を持つ実用的なセキュリティコンサルタントから、暗号化または仮想化の複雑な詳細の博士号を持つ人々にまで及びます。  
 他の開発者と対話します。  
27.6.5ワークフローの進化  
専門の開発チームは、適切なツールを使用して、この本で説明されている多くの問題を回避します。  
よく管理されたライブラリを使用することで、タイミング攻撃や弱い乱数ジェネレータなどの暗号の問題を回避できます。  
たとえば、Coverityの場合、作成者は、プロジェクトの最初から使用するのは素晴らしいことですが、途中でそれを採用すると、突発的な20,000のバグレポートと出荷日が突然発生するため、実際のコストがかかると説明しています。数ヶ月遅れます[235]。  
暗号化ライブラリの場合、第6章で、デフォルトとしてECBなどの弱い動作モードを提供する傾向があるため、代わりにチームがGCMを使用するようにする必要があります。  
 ウェブサイトを更新する際のクロスサイトスクリプティングの脆弱性やSQLインジェクションを回避するため、またはクライアントデータをS3バケットで誰もが読み取れる状態にしないようにするために、常に新しいツールを追加します。  
ただし、すべてを購入することはできません。セキュリティ業界には、無知な顧客を利用する悪質なオペレーターがたくさんいます。  
独自の自動化を行うと、スタッフの能力を高め、時間を節約できます。  
ここで重要なことの1つは、APIを安全に進化させることができる必要があるということです。  
理想的な世界では、タイプセーフティメカニズムを使用してAPIの問題を防止するためにプログラミング言語を利用することになります。  
セクション20.5で、暗号化ハードウェアセキュリティモジュールのAPIが拡張され、何百もの銀行のレガシーATMシステムをサポートするようになったことを確認しました。  
これを管理するより良い方法があるはずです。ファンアウトの高いアプリケーションは、共通のコアとさまざまな呼び出し側アプリケーションのプラグイン可能なプロキシを備えたマイクロサービスアーキテクチャの方向に移動すると思います。  
27.6.6そして最後に...  
8 HRの主な仕事は損傷の制限であり、退職者があなたを訴えることを阻止します。  
また、内向的すぎて他の人を惹きつけたりやる気を起こさせたりすることのできないエンジニアによってもうまくいくことができません。  
テクノロジーの成功するマネージャーは、テクノロジーを愛し、理解する必要があります。彼らはまた人々を愛し、理解しなければなりません。  
彼らは、R＆Dラボで最も生産性の高いイノベーターになる可能性があります。  
彼らはあなたのエンジニアリングスタッフに道徳的なリーダーシップを提供し、あなたの顧客に安心を与えるかもしれません。  
しかし、鍵となるのはこれだと思います。つまり、社内に1つ以上のエンジニアリング専門職がいるということです。  
 誰が彼らを導くのですか？  
 それらをどのように成長させ、開発しますか？  
   
セキュリティ、安全性、またはその両方の重要な要件を満たさなければならないシステムを構築または強化するためにプロジェクトを管理することは難しい問題です。  
言い換えると、ソフトウェアのセキュリティが必要です。他の機能や、安全性やリアルタイムパフォーマンスなどのその他の新しいプロパティも必要です。  
より穏やかなシステムの進化には、要件に対する微妙な変更が伴います。  
この動的なコンテキストのため、安全なコードを書くのは困難です。最初の問題は、何をしようとしているのかを理解することです。  
適切な人材の採用、適切なツールの提供、適切な作業方法の開発の支援、適切な方法での専門知識によるバックアップ、そして何よりも、適切な人材を育成するために働く環境の構築には、多くの課題があります。セキュリティ機能。  
ただし、ソフトウェアエンジニアリング、応用心理学、経済学、管理との境界にあるため、ほとんど注目されていません。  
セキュリティ経済学とセキュリティ心理学はここ数年で大きな進歩を遂げており、開発者がセキュリティツールをより使いやすくするために、さらに多くの作業を行う必要があることがわかりました。  
しかし、多くの失敗は組織の行動によるものです。  
誰かが組織の不十分なインセンティブによって引き起こされたセキュリティ障害のケース履歴のライブラリを収集する場合に役立ちます。たとえば、[876、？きちんとした経験的基盤があれば、何が続くでしょうか？  
 脅威環境の変化を監視し、システム開発だけでなく内部統制などの支援活動にそれらをフィードするための制度的構造をどのように設定するのでしょうか。  
 スタッフが適切なレベルの注意を払って行動するという意味で「安全インセンティブ互換」である組織をどのように設計できますか？  
 前の章で、アルカイダによってもたらされた明らかに新しい脅威に対する政府の対応がさまざまな点で不適応であったことを確認しました。パンデミックなどの社会的リスク。  
その結果、セキュリティ開発の取り組みの多くは、セキュリティと安全性のリスクを適切に管理するのではなく、コンプライアンスに向けられています。  
 それとも、ビッグフォー会計事務所の解体など、より広範な構造変化が必要ですか？  
フレッドブルックスの「神話の男月間」[328]やナンシーレベソンの「セーフウェア」[1149]など、誰もが読むべき古典があります。  
2020年の時点で誰もが読むべき最新の本は、おそらくSRE [236]と「安全で信頼できるシステムの構築」[23]に関するGoogleの本でしょう。  
安全機能と完全性の要件、および関連するエンジニアリングプロセスに関する最も一般的な標準セットはIEC 61508です。業界固有の標準のセットがさらにあります。  
航空業界では、電子ハードウェアの場合はRTCA DO-254、ソフトウェアの場合はRTCA DO-178Cですが、モーター業界では、安全性についてはISO 26262、セキュリティについてはISO 21434です。ただし、これはまだ草案です。  
他の工学分野から多くを学ぶことができます。  
ITプロジェクトの失敗は、もう1つの必要な調査対象です。 Christoph Loch、Arnoud DeMeyer、Michael Pichによるプロジェクトの不確実性を管理する方法に関するケースブックがあります[1179]。  
ビジネススクールの文献では、組織的な側面について詳細に説明していますが、これは部外者を困惑させる可能性があります。  
1冊の本しか読まない場合は、ルイスピノーの「コンサルティングデーモン」にしてください。これは、大手コンサルティング会社が顧客を引き裂く方法についての元インサイダーの告白です[1527]。  
誰が誰にアクセスできるかだけでなく、誰が誰に耳を​​傾ける用意があるか、誰が誰からの命令を無視するかだけです。