彼らはセキュリティの第一人者である必要はありませんが、彼らが何を構築しようとしているか、それがどのように機能するかを理解する必要があります。  
–Geo↵Nunberg-キツネは多くのことを知っています。ハリネズミは大きなものです。  
1はじめにこれまで、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、その瞬間の情熱を信仰します。  
 1960年代に、フレッド・ブルックスレッドは、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でチームを率いました。  
 86827を作る魔法の数式はありません。  
 リスク管理本質的に難しい仕事は簡単です。  
 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、偶然であれ氷の結果であれ、何がうまくいかないのかについて体系的に考えることを伴います。  
 このコースは、同僚のAlastair Beresfordが2019年にサバティカルに在籍していたときに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、定義からプロトコルと暗号の基本に至るまでの基本を紹介し、ケースヒストリで示されているように、技術的な問題だけでなく、人的および組織的な問題の重要性を紹介します。  
 成功とは、態度や仕事のやり方、スキルです。  
 次に、組織の行動の結果として発生するいくつかの危険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。  
2リスク管理安全エンジニアリングとセキュリティエンジニアリングの両方の中心にあるのは、優先順位に関する決定です。何に対する保護にどれだけ費やすかです。  
 それはしばしばひどく行われます。  
 台湾や韓国など、SARSやMERSの最近の経験を持つ国は、居住者をテストして大規模な連絡先を追跡する準備ができており、迅速に対応する準備ができていました。  
では、実際にリスクレジスタとは何でしょうか。 Security Engineering869Ross Anderson27で使用される一般的な方法論。  
 私の大学の管理機関であるリスク管理は、問題が発生する可能性のあるもののリストを作成し、深刻度と発生確率のスコアを1から5にして、これらを掛け合わせて1から25の数値を取得します。  
 次に、これらの各リスクを軽減するために講じた措置を書き留め、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て、それぞれの所有者になるように1つのシニアOcerを割り当てます。  
。  
）それが殺すかもしれない人々の数によって（何百万？何千？何十？）そしてそれからあなたが各世紀にどれだけ期待するかによって確率のためにそれを評価します。  
 最も深刻なのは、パンデミック・インフルエンザ、国民の停電、深刻な洪水です[361]。  
 あなたは費用便益分析を行い、優先順位を政策に変えます。  
 国家安全保障理事会が安全保障局と諜報機関によって捕らえられたので、英国はパンデミックをほとんど無視しました。彼らはテロを優先し、保健長官は常連ではなかった[1848]。  
3;ここでは、失敗の別の側面がポリシーのオーバーシュートであったことを追加します。  
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、予想される損失に平均年に予想されるインシデントの数を掛けたものとして、考えられる損失シナリオごとに年間損失期待値（ALE）を計算することです。  
1。  
ALEは、米国政府調達で使用する手法として、NISTによって長い間標準化されてきました。  
2。  
005 $ 250,000ATM詐欺（大）$ 250,000  
5 $ 10,000テラーが現金を受け取る$ 3,240200 $ 648,000図27。  
 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。  
 次に、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、当て推量にどの部分があり、政治にどの部分があるかを理解する必要があります。  
 さまざまな産業は、それらが進化した方法と規制の歴史のために、異なって物事を行います。  
EuropeanUnionは、ブリュッセルよりも安全性に関心が低く、複数の製品を用意するよりもEUの安全仕様に合わせてOEMが設計する方が簡単であるため、最大の市場である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、医療機器用のソフトウェアは、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、保険は循環型産業であり、2017年以降、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制限されています。セクション28で、彼らが何をするかのメカニズムについてさらに説明します。  
9。  
 （コロナウイルス危機は、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しているため、相関リスクについて企業に教えています。流行のせいで停滞のリスクがオフィスに到達できないというリスクについて明示的に言及している場合でも、企業は保険会社に順番に尋ねています保険のポイントは何ですか。  
3。  
 取り組まれているリスクは、表面上は機能しているよう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、過失請求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。  
これは、セキュリティ予算を正当化するために使用される言説に波及します。  
80年代後半から、ウイルスが企業の想像力を乗っ取り、人々はウイルス対策ソフトウェアを販売するようになりました。  
 1990年代後半には、PKIが熱狂しました。  
このすべてのフープの中で、セキュリティの専門家は本当の脅威が何であるかを理解するために一定の頭を保ち、努力する必要があります。  
 まず、安全工学から何を学べるか見てみましょう。  
3安全が重要なシステムからの教訓重要なコンピュータシステムとは、特定のクラスの障害を可能な限り回避する必要があるシステムです。  
安全上重要な種類の明らかな例には、飛行制御や自動ブレーキシステムなどがあります。  
27。  
1安全工学の方法論古典的なセキュリティ工学のような安全工学の方法論は、安全分析から仕様、そして製品に至るまで体系的に機能する傾向があり、改造しようとするのではなく、最初から安全を構築していると想定します。  
。  
）;ハードウェアおよびソフトウェアコンポーネントの危険性を追跡します。これにより、重要であると識別されます。重要なオペレーターの手順を特定し、さまざまな応用心理学およびオペレーションズリサーチの問題を研究します。安全メカニズムが何をしなければならないかを指定する安全機能要件と、安全機能が満足に実行される可能性を指定する安全整合性要件を設定します。最後にテスト計画を決定します。  
 基本的なフレームワークは、化学プラントの制御システムなどの比較的単純なプログラム可能な電子機器のための基本的な安全フレームワークであるISO 61508などの標準で定められています。  
この安全事件は、何かがうまくいかなかった場合に、その証拠となるセキュリティ工学872ロス・アンダーソン27を提供します。  
 安全上重要なシステムからの教訓•�••�•�•�M�（a）�（b）�M�図27。  
これは通常、ハザード分析、安全機能と完全性の要件、および必要な故障率が達成されたことを示すテスト（コンポーネントレベルとシステムレベルの両方）の結果で構成されます。  
車両はサプライチェーンがあるため、より複雑なケースです。  
 次に、元の機器メーカー（OEM）がいます。自動車の場合、通常は同じ会社ですが、常にそうであるとは限りません。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、部品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）。また、緊張もあります。すでに述べたように、安全性の認定は、セキュリティパッチの適時の適用を妨げることがあります。  
27。  
2ハザード分析理想的なケースでは、システムからハザードを完全に設計できる可能性があります。  
2。  
 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なくショートします。  
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。  
3。  
3。  
3：–脅威はツリーバンクに直接影響を与えるため、SWIFTには有効なトランザクションを偽造する手段がなく、そのステータスとシステムの信頼性を低くする必要がありました。  
 同じことがプライバシーエンジニアリングにも当てはまります。  
 ただし、これには明らかなプライバシーの危険があり、代わりに全員の携帯電話でBluetoothの連絡先履歴を保持し、病気で電話をかけた人の連絡先履歴をアップロードすることで軽減できます。  
27。  
3フォールトツリーと脅威ツリーできるだけ多くのハザードを排除したら、次のステップは、事故を引き起こす可能性のある障害を特定することです。  
 このトップダウンアプローチは、避けなければならない少数のよく知られた悪い結果を伴う複雑なシステムがある場合に自然です。  
 図27。  
脅威ツリーは米国国防総省で使用されています。  
 次に、各先行条件を追加して再帰します。  
 これから得られるもう1つの優れた点は、攻撃パス間の共通性を視覚化することです。これにより、最も少ない攻撃で最も多くの攻撃を中断する方法を簡単に推論できます。  
3。  
 脅威ツリーはシステムの攻撃マニュアルに相当する可能性があるため、高度に分類される可能性がありますが、DoDの要件です。システム評価者または認定者が重大な追加の攻撃を見つけることができる場合、製品は失敗する可能性があります。  
3。  
 これには、システムの各コンポーネントの障害による影響を、ミッションへの影響まで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ときに空港まで滑空できない海や山の上を飛行する場合、エンジン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、保守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。  
致命的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年の喪失です。  
 冷気によってもろくなったOリングが故障し、シャトルと7人の乗員が失われた。  
 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す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ている場合、工学だけでなく法律と経済についても考える必要があります。  
27。  
5脅威のモデリングフォールトツリーとFMEAはどちらも、システムを十分に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。  
セキュリティエンジニアリング875ロスアンダーソン27。  
 安全批評システムからの教訓は、人々が時間をかけて学んだアプリケーションです。  
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相互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、ハードウェア障害とソフトウェア障害の障害ツリーを構築します。  
 この分析は、キャビンCANバスとpowertrainCAN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、内部関係者について体系的に考えなければならないことを意味します。  
 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。 FMEAを使用している場合は、悪意のある「ランプオ」メッセージが表示されないように、さまざまな場所に対戦相手を追加できます。  
これは、純粋にトップダウンまたはボトムアップというよりは、中間的なアプローチです。  
 次に、1つのモジュールから別のモジュールへの、システム全体の攻撃パスを追跡します。  
 脅威モデルは、アーキテクチャのレビューからターゲットコードのレビュー、侵入テストまで、セキュリティ開発ライフサイクルのさまざまなポイントでさまざまな目的に使用できます。基本的なアプローチは、安全メカニズムに対するハザードのマトリックスを構築することであり、安全ポリシーが少なくとも2つの独立したメカニズムによって重大なハザードを抑制しなければならない場合は、関連する各列の2つのエントリを確認できます。  
 別のアプローチであるシステム理論プロセス分析（STPA）は、ハザードから始まり、次に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回避するのに役立ちます[1150]。  
 複雑さを高めるために何らかの方法でセキュリティエンジニアリング876ロスアンダーソン27。  
 安全性の重要なシステムからの教訓管理可能な場合、安全とセキュリティの目標の階層を整理する必要があるかもしれません。  
 この階層により、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。  
3。  
 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、さまざまな種類のアクティビティでオペレーター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。  
 同じレッスンが次々と起こった航空事故調査から出てきました。  
 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、核反応を抑制し、航空機をまっすぐ水平に戻し、自律走行車を道路脇で停止させます。  
 芝生の端があると、車両は道路のどちら側にい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で詳しく説明します）。  
4）飛行制御コンピュータは飛行機のレベルを維持しようとしましたが、欠陥のある迎え角センサーによって混乱し、代わりに飛行機を地面に沈めました。  
 昔は、各フィードは1つのゲージまたはダイアルに送られ、それらのためのスペースはあまりありませんでした。  
 昔は、設計者は緊急時にパイロットにトンネルのビジョンを与えることで、本当に必要な6つの計器を真ん中に置くことを知っていました。  
 それは航空よりはるかに広いです。  
 ステアリングコントロールが誤ったヘルムステーションにシフトし、エンジンが時間内にスロットルされなかったため、忙しい船積みのレーンを横切ってポートへの命令されていない方向転換が行われました。セキュリティエンジニアリング877ロスアンダーソン27。  
 安全批評システムからの教訓ケミカルタンカーとの衝突、および10人の船員の死[1929]。  
 特定のタイプのエラーをより一般的にする認知バイアスやその他の心理的要因については、かなり多くのことが知られています。それらについては第3章で説明しました。賢明なエンジニアは、フィールドでどのように機能するかを研究します。  
 これにより、オッズがわかっているリスクの領域から、そしてオッズがわからない不確実性の領域へと私たちを連れ出し始めます。  
 セキュリティの使いやすさは、エンドユーザーに直感的なインターフェースを事前に提示することだけではありません。  
各重要なタスクを実行する人々のスキルレベルと、エラーの可能性に関する既知の推定値について、現実的になることが重要です。  
 車は、年老いて虚弱で、若くて経験の浅い、乗客の注意をそらす、またはアルコールの影響下にあるドライバーによって操作される可能性があります。  
 彼らは、どのようなストレスの組み合わせが致命的な事故につながるのか、そしてコックピットのタイプがどのように変化するかを観察します。  
 運航、安全運航のためのインセンティブは十分に強力であり、適切に調整されており、規模は学習システムをサポートするのに十分な大きさです。  
 その結果、ソフトウェアは航空機が装備していた2つのセンサーを使用するのではなく、単一の迎角センサーに依存し、センサーの故障は致命的な事故につながりました2。  
操縦席のシステムの1つが動作していない状態で、旅客機を飛行させたことにより、いくつかの空中墜落が発生しました。パイロットは、コックピットディスプレイへのデータフィードの1つが信頼できないことを知的に認識しているかもしれませんが、他の機器でチェックするのではなく、反射によって圧力下でそれに依存している場合があります。セキュリティエンジニアリング878ロスアンダーソン27。  
 保護の優先順位付けの目標直接的な影響が軽減されても、目に見えてテスト可能です。  
 安全ケースが分離している場合、承認後にそれを回避し、適切に維持することができません。  
）プロジェクトベースのソフトウェア管理から俊敏な方法論への移行、およびDevOpsを介したDevSecOpsへの移行により、最終的にセキュリティ管理が製品の進化に組み込まれ始めています。  
最後に、安全性はセキュリティに似ており、後付けするのではなく、システムの開発時に組み込む必要があります。  
 安全性はランダムな障害の影響を扱いますが、不安は、システムのコンポーネントのいくつかを、都合の悪いときに最も損傷を与える可能性のある方法で失敗させる敵対的な敵を想定しています。  
4。  
効果的には、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。  
これは、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]。  
4保護目標の優先順位付け完全に新しい製品を作成するプロジェクトがある場合、または既存の製品を根本的に変更するプロジェクトの場合、最初の原則から保護優先順位を検討するのに時間をかけることが考えられます。  
 セーフティケースまたはセキュリティポリシーを詳細に開発する場合、コンテキストを理解することが不可欠であり、この本の多くは、幅広いアプリケーションに関連する脅威モデルに関するものでした。  
ビジネスシステムの場合、分析はリスクと報酬のトレードオフにかかっています。  
 会社の売上高が1,000万ドル、総利益が100万ドル、盗難損失が150,000ドルの場合、「盗難を防ぐことで利益を15％増やす方法」について損失削減のピッチを作ることができます。しかし、売上高を2千万ドルに2倍にできれば、株主はそれが損失を350,000ドルに3倍にしてもそれを好むでしょう。  
55m、15％ではなく85％上昇。  
 詐欺管理戦略についてaSecurity Engineering879Ross Anderson27と話し合うとき。  
 保護目標の優先順位付け小売業者の数、私は、最良の結果を得た企業は、詐欺管理チームが財務ではなく販売に報告した企業であることに気づきました。  
 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、チーフマーケティング担当者の目を照らす見込みです。  
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ことを学びました。  
時間の9％が微細になります。 99に達するために何百万も費やす意味はありません。  
意図的に0を設定する方がよいでしょう。  
これは私に開かれた議論の安全管理の1つをもたらします：依存するソフトウェアのいずれかでCVEを開かないことを目的とすべきですか？チェックボックスのアプローチは「もちろんオープンCVEで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、セクション27で詳しく説明します。  
7。  
したがって、「セキュリティの強化」についてのみ話すことができる人々を信用しないでください。  
 この本の最初の版では、スーパーマーケットでのセルフサービスチェックアウトのケーススタディを紹介しました。  
 一部は損失にこだわり始め、製品の重量について顧客に積極的に挑戦することでセキュリティがユーザビリティの邪魔になりました。  
最終的に業界はセルフチェックアウトレーンの運用方法を考案しましたが、実装の品質は依然として大幅に異なります。  
アマゾンはまた、完全にセルフサービスの食料品店を試していました。  
食料品店がすべての商品をカウンターの後ろに置いていた当時のセルフサービスのスーパーマーケット自体から始まって、多くの現代のビジネスモデルはかつては危険すぎると考えられていました。  
 ビジネスでは、利益はリスクに対する報酬です。  
最近の1つの例は、ビデオ会議プラットフォームZoomです。これは、2020年3月に2000万人のユーザーから2億人に成長し、その過程でエンタープライズプラットフォームから公共ユーティリティのようなものに変化し、セキュリティエンジニアリング880Ross Anderson27を強制しました。  
 方法論主要なセキュリティエンジニアリングの開始[1763]。  
 論理的には、発展途上国での人間の生活の価値は数百万ドルで、平均的な人の生涯収入です。健康政策の差異はさらに大きく、救命救済の費用はアフ・ジャブといくつかの癌検診では数百ドルから、最も効果的でない介入では数十億に及ぶ[1869]。他の安全の状況では、国内の煙警報器は、命を救うために数百ドルの費用がかかりますが、「テロとの戦い」の数は数十億です[1350]。  
2。  
3。  
 安全の選好は、敵対的な行動の脅威によって非常に急激に変化する可能性があります。人々は、デバイスがハッキングされる可能性があり、その時点で1,000万分の1のリスクでさえ怖くなるまで、設計が不十分な医療機器によって殺される10,000分の1のリスクを完全に無視する可能性があります。  
4。  
5方法論ソフトウェアプロジェクトは通常、計画よりも時間がかかり、予算よりもコストがかかり、予想よりも多くのバグがあります3。  
とにかく、ソフトウェアエンジニアリングという用語は1968年にブライアンランドールによって提案され、次のように定義されました。ソフトウェアエンジニアリングは、信頼性が高く、実際のマシンで効率的に動作するソフトウェアを経済的に取得するためのサウンドエンジニアリングの原則の確立と使用です。  
 それ以来、多くの進歩がありましたが、結果は予想外でした。  
 しかし、大規模プロジェクトの約30％が失敗していることがわかります。違いは、失敗がはるかに大きいことです。  
これについては、この章の最後にある組織行動に関するセクションで詳しく説明します。  
 一部の初期のマシンがサポートしていたアセンブリ言語など、不適切なツールを使用したプログラミングには、付随する複雑さが伴います。このような言語でグラフィカルユーザーインターフェイスを使用して最新のアプリケーションをプログラミングするのは、退屈な作業であり、エラーが発生しやすくなります。  
セキュリティエンジニアリング881ロスアンダーソン27。  
 方法論はまた、大きく複雑な問題を処理する際の本質的な複雑さでもあります。  
偶発的な複雑さは、主に技術ツールを使用して処理されます。  
 彼らは自分の費用をもたらします。多くの脆弱性はC言語の特性の結果であり、履歴を再実行している場合は、代わりにRustのようなものを使用します。  
本質的な複雑さは微妙に異なる何かを必要とします：問題を管理可能なサブ問題に分割し、これらのサブ問題が相互作用できる範囲を制限するのに役立つ方法論。  
 基本的に2つのアプローチがあります–トップダウンと反復です。  
5。  
 システムの要件についての簡潔な説明から始めるという考え方です。これを仕様に詳しく説明します。システムのコンポーネントを実装およびテストする。次にそれらを統合し、システムとしてテストします。その後、システムを展開してliveoperationを行います。  
 私が1980年代に銀行で働いたとき、それはそこでも承認されたプロセスであり、IBMによって、政府によって、そして大規模な会計事務所によって徹底的に推進されました。  
 このチェーンの最初の2つのステップでは、適切なシステムを再構築しているか（検証）、次の2つのステップでは、適切に再構築しているか（検証）に関するフィードバックがあります。  
 しかし、それはそうです。  
 各ステージから前任者へのフィードバックはありますが、（たとえば）システムテストから要件へのシステムレベルのフィードバックはありません。  
 これはセキュリティエンジニアリング882Ross Anderson27です。  
 METHODOLOGY要件-検証-仕様-検証-実装-＆単体テスト-検証-統合と-システムテスト-検証-コード-ビルド-フィールド-運用-保守-図27。  
 しかし、それはトップダウンではなく左から右に書かれていますが、それは要件がシステムを駆動する一方向のプロセスであり、受け入れテストは、経験に照らして要件を進化させるメカニズムではなく、要件が満たされていることを確認します。  
ウォーターフォールモデルは、第二次世界大戦直後にドイツのGerhardPahlとWolfgang Beitzが工作機械などの機械設備の設計と構築のために開発した方法論の先駆けでした[1490]。どうやらPahlの学生の1人は後に、経験豊富なデザイナーが実際に行うことの正確な説明としてではなく、エンジニアリングの学生を始める手段として最初に設計されたと後で述べました。  
ウォーターフォールモデルの長所は、システムの目標、アーキテクチャ、およびインターフェイスの早期の明確化を強制することです。目標とする明確なマイルストーンを提供することにより、プロジェクトマネージャーのタスクを容易にします。各ステップごとに個別の料金を設定できるようにすることで、コストの透明性が高まり、仕様の変更が遅れる場合があります。幅広いツールと互換性があります。重要な問題は、開発またはプロトタイピング作業の前に要件が詳細に知られているかどうかです。  
 惑星間宇宙探査機のように1発しか得られないような外部的な理由から、トップダウン方式が必要になる場合があります。  
5。  
 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るにつれて、すべてがますます複雑になり、多くの企業の現実は「メンテナンスが製品」であるということです。  
 それによって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。  
27。  
2反復的なデザイン：スパイラルからアジャイルまで反復的な開発には、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強化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。  
 最初のプロトタイプが重要な最初のステップである多くのアプリケーションがあります。デモを投資家に紹介することを目的とした新興企業から、ショーワフォーカスグループへの新製品のモックアップを構築する会社を通じて、提案された技術が完全に不可能ではないことを確立することを目的としたDARPA苗プロジェクトまで  
 重要なのは、直面している最悪の問題を解決して、プロジェクトのリスクをできるだけ減らすことです。  
5。  
5：–スローガンのスパイラルモデルアップ：「最悪の問題を解決します。  
進化的アプローチの初期の提唱者はHarlan Millsで、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と教えていました。  
 確かに、MicrosoftはWordの書き換えを何度も試みましたが、毎回あきらめました。  
 この世界観では、製品はプロジェクトの結果ではなく、以前のバージョンの継続的な変更を伴うプロセスの結果です。  
 （IBMersは、バグの多い信頼性のないコードを生成した規律のないハッカーの集まりとしてMicrosoftを非難しました。しかし、Microsoftが主要なビジネス市場を盗んだ後のIBMの臨死体験は、IBMの開発へのアプローチの厳格さに起因するとされています[390]。  
進化の発展に関する重要な洞察は、生物種の各世代が種を存続させるために存続可能でなければならないのと同様に、進化するソフトウェア製品の各世代も存続可能でなければならないということです。  
 定期的（通常は1日に1回）に、製品のさまざまな機能に取り組んでいるすべてのチームがコードをチェックし、ビルドにコンパイルされて、大量の入力に対して自動的にテストされます。  
 誰かがコードによってビルドを壊した可能性は常にあります。そのため、現在の「世代」は、機能した最後のビルドであると見なします。  
 最終的にはSecurity Engineering885Ross Anderson27になります。  
 方法論ビルドとテスト体制の階層。  
テストの技術は、1990年代から2000年代初頭のソフトウェアエンジニアリングにおける最大の実用的な改善であったと考えられます。  
 進化論的発展への動きは、他の多くの変化と関連していた。  
 プログラマーはバグの多いコードをフェンスの上に投げて、誰か他の人がそれを修正することを期待できるため、インセンティブは適切ではありませんでした。  
マイクロソフトは、アナリスト、プログラマー、テスターの区別を廃止しました。開発者だけが顧客に話しかけ、自分のバグの修正も担当していました。  
 Steve Maguire氏によると、これがMicrosoftが32ビットオペレーティングシステムの世界を支配するための戦いに勝利した理由です。より優れた開発方法論により、IBMから1000億ドル規模のビジネスソフトウェア市場を獲得することができました[1209]。  
5。  
 サーバーはLinuxに移行し、個人ユーザーがMacを購入し始めました。彼らの内部トレーニング資料は、より広範なエコシステムの変化を推進するのに役立つ本や紙になりました。  
3。  
 時間の経過とともにさらに登場し、そのセキュリティ開発ライフサイクル（SDL）は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を通じて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。  
5。  
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。  
次に、成熟度モデルがあります。  
 十数人の有能なミュージシャンを集めるだけでなく、ソフトウェアにも同じことが言えます。  
 CMUの調査によると、新たに形成されたチームはプロジェクトの作業量を過小評価する傾向があり、彼らが要した時間にも大きなばらつきがありました。一緒にベストを尽くしたチームは、平均的な開発時間の経過時間を予測することができましたが、分散も減少しました[1937]。  
 マイクロソフトはこれを採用し、開発者チーム向けに4レベルのセキュリティ成熟度を定義しています。  
5。  
 当時のセキュリティエンジニアリングアプローチの多くはウォーターフォールの想定に結び付けられており、それ自体での自動テストは、多くの理由でセキュリティエンジニアにとってあまり役に立ちません。  
 特定の攻撃タイプは特定の救済策を必要とすることが多く、仕様エラーがユーザーのインターフェース機能と相互作用する場合など、多くのセキュリティフローがシステムの抽象化レベルを超えます。これは、自動テストを考案することが難しい種類の問題です。  
 セキュリティエンジニアリング887ロスアンダーソン27の場合、これは特に重要です。  
 方法論開発スプリントは、相互にやり取りできる多くの機能を追加します。  
 これは、安全要件またはセキュリティ要件を持つシステム全体でかなり一般的です。  
多くの無視されている問題は、セキュリティ要件が進化し、維持およびアップグレードする必要があることです。  
一部の変更は暗黙的です。たとえば、静的分析ツールをアップグレードすると、既存のコードベースに何百もの「新しい」バグが見つかる可能性があります。  
Windowsに脆弱性が見つかった場合、パッチを適用するだけでは不十分です。作成した人が、コードベース全体に散在する同様の脆弱性を12個作成した可能性があります。パッチを公開すると、悪意のある人がそれを調査して理解します。  
 マイクロソフトとその顧客の両方のコストを管理するために、同社は2003年に、パッチを1つにまとめて月刊アップデート（現在は有名な「patchTuesday」）にまとめ始めました。  
 そして、そのようなパッチはさらなる依存関係を作成します。  
 しかし、多くはそうではありません。スマートフォンとデスクトップの両方のアプリの70％は、使用するオープンソースライブラリに脆弱性があり、通常、simpleupdate [1695]で修正できます。  
安全システムと多くの点で類似している安全重視システムにも、ほぼ同じ考慮事項が適用されます。  
 安全性は、ほとんどのアプリケーションで、広範な市販前テストに依存していたものです。  
 それでも、最新バージョンのセーフティクリティカルシステムがセーフティケースを満たすことを保証するには、大規模で高価なテストが必要になる場合があります。  
自動車業界の企業は相互に不審であり、NDAの下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。  
 これらは実際のお金がかかり、実際の車両を維持する必要もあります4これもまた問題を引き起こします：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ていたときに、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て実験を実証しようとしていたところ、突然キャプチャソフトウェアは機能しなくなります。  
セキュリティエンジニアリング888ロスアンダーソン27。  
 方法ロードテスト。  
これは、Tesla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、レガシーカー会社に数週間かかる変更を数週間でテストして出荷できます[404]。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。  
27。  
5サービスとしてのソフトウェア2010年代初頭以降、販売されてユーザーに配布されるのではなく、セントラルサーバーでホストされ、シンクライアントによってアクセスされ、サブスクリプションベースで支払われるソフトウェアが増えています。  
 ベンダーがこれらのタスクをすべての顧客から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は、継続的な統合と継続的な展開です。  
 ステージング環境と呼ばれる実際の顧客データのスナップショットに対してドライランでテストできるため、アップグレードははるかに制御しやすくなります。  
展開自体は暫定的なものです。  
個別のインスタンスは、堅牢性を向上させるために個別の障害ドメインも提供します。  
 カナリアが存続する場合、デプロイメントは新しいサービスインスタンスに段階的にロールフォワードできます。  
 ユーザーがトランザクション間でデザインの古いバージョンと新しいバージョンを切り替えても問題が発生しないように注意する必要があります。  
セキュリティエンジニアリング889ロスアンダーソン27。  
 方法論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、ユーザーごとのセッション時間、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はかなり小さいので、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。  
 システムの機能拡張を展開しながら、複数の同時実験を実行することについての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、迅速にレプリケートできます。  
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がそれ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「Site Reliability Engi-neering」です。 SREは、DevOpsの用語です[236]。  
 彼らは2000年代から2010年代初頭にかけて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、優れた能力を発揮しました。  
 コアストラテジーは、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、急速な革新と可用性のバランスをとることです。  
Security Engineering890Ross Andersonの場合、9999％の可用性27。  
 METHODOLOGYISPは、ユーザーがサーバーに99％または99しかアクセスできないようにします。  
 現実的なエラーバジェットを設定する場合は、0と言います。  
01％利用できない場合、それを使用して多くのことを達成できます。  
エラー予算の残りの部分を使用して、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押し出します。  
）セクション12。  
6。  
 WardCunninghamによるこの概念は、開発のショートカットが好まれているという観察を要約しています。  
 技術的負債は、新興企業、またはその寿命が近づいているシステムにとっては理にかなっているかもしれませんが、多くの場合、不十分な管理または不十分なインセンティブの産物です。  
 銀行がコアバンキングシステムを交換しなければならないことは、非常に高価で破壊的です。  
 DevOpsの哲学の1つの重要な側面は、無借金で運営することです。  
5。  
 理論的には、これは「すべてをコードとして」という戦略を意味します。実際には、既存のセキュリティ評価（および関連する場合は安全性の事例）を維持するだけでなく、新しい脅威、環境の変化、および驚くべき脆弱性に対応することを意味します。たとえば、セクション12で言及しました。  
2 DevOpsは、銀行が長年にわたって依存してきた開発と生産の分離を弱体化させています。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、2つの例を挙げます。MicrosoftworldとGoogleワールドと大まかに呼ぶことがあります。  
27。  
6。  
 一般的なクライアントには、オンプレミスシステムとクラウドシステムが混在しており、主に前者から後者に移行する新しい開発があります。  
5。  
 脅威のモデリング、静的分析、動的分析、ファズテスト、アプリとネットワークの監視、セキュリティオーケストレーションとインシデントレスポンスを実行するために使用される数十のツールは、1つのツールから別のツールにデータをコピーする何十人もの人に大きなオーバーヘッドを課します。  
このエコシステムをサポートするために、MicrosoftはSDLを次のステップで拡張しました。脅威モデリング;暗号レイフィ標準;サードパーティコンポーネントのセキュリティリスクの管理。侵入テスト;そして標準化されたインシデント対応。  
 有能なシステム統合および施設管理会社は、これらのステップをワークフローに組み込む方法を考え出しました。実際の作業の多くは、彼らまたは彼らの顧客が購入したサードパーティのセキュリティ製品を統合することを含みます。  
そのような会社におけるDevSecOpsの組織化の原則は、「シフトレフト」を実行することです。これにより、多くのことをカバーできます。統一テーマは、ソフトウェアやインフラストラクチャなどのセキュリティをコードベースに移動することです。  
 2010年には、マイクロソフトは運用上のセキュリティをソフトウェアのセキュリティとは別のものと見なしていましたが、最新のAzureショップでは、継続的な監視、手動の侵入テスト、そして最後に何か問題を発見したサードパーティのバグ報奨金を使用して、展開を追跡します。  
27。  
6。  
 DevSecOps戦略は、Amazonでもやや似ていますが、自社製品向けに最適化されています。 [1966]のCTO WernerVogelsによって説明されています。  
 このセクションは、彼らの本と、最近主要なサービス会社で働いてきた同僚を利用しています。  
 したがって、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、エラーを減らすのに役立つかもしれませんが、重点はセキュリティエンジニアリング892ロスアンダーソン27です。  
 主要なサービス企業の方法論は信頼性です。  
9％の可用性、そして0の残留エラーバジェットを使用します。  
これにより、回復可能性の設計、理解可能性の向上のための設計、人間が可能な限り生産システムに触れないようにするという願望など、さらなる原則が推進されます。  
2。  
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。  
 何千人ものユーザーから数十億人にすべてをスケールアップすることは、もっと多くのものが自動化されなければならないことを意味します。  
 サイトの信頼性エンジニアリングでは、アラームはできるだけ単純で、予測可能で、信頼できるものでなければなりません。しかし、セキュリティの面では、ランダム化を行うことをお勧めします。  
 各コンポーネントは通常、多数の並列コピーまたはシャードとして実装され、さらに小さな障害ドメインを提供します。  
 コンパートメント化されたシステムは、回復力のために設計することもできますが、これは簡単ではありません。  
Webページ用の標準フレームワークもあり、そもそもSQLインジェクションやクロスサイトスクリプティング攻撃を防ぐだけでなく、さまざまな言語のサポートも提供します。  
 単一のフロントエンドは、負荷分散とDDoS保護のための単一の場所を提供することもできます。セキュリティエンジニアリング893ロスアンダーソン27。  
 方法論数十の異なる言語のサポートなど、他の多くの機能と同様。  
 理解しやすい、secure-by-constructionAPI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。  
27。  
6。  
 Dev、Sec、およびOpsconvergeとして、メトリックと管理プロセスは、セクション21で説明したネットワーク防御メカニズムと収束します。  
 しかし、これはすべてインテリジェントに管理する必要があります。  
 不適切に実行された企業は、メトリックを管理し、緊張を生み出します。セキュリティ状態は、悪者を締め出すという矛盾した目標になり、チーム自身の正当化に使用されるすべてのメトリックをヒットすることによって、「獣に餌をやる」ことになります。存在[1846]。ただし、どちらの場合でも大きな要因の1つは、脆弱性のライフサイクルです。  
27。  
7脆弱性サイクル1970年代と1980年代に戻ると、人々は時々、システムのセキュリティバグを発見し、それらを却下して侵入とパッチとして修正するという進化的な手順を説明しました。  
 これまで見てきたように、暗号化機器などのいくつかのエッジケースを除いて、それは実際には機能しませんでした。  
脆弱性サイクルは、再検索者である誰かがベンダーによって維持されているシステムの脆弱性を発見するプロセスで構成されています。  
5。  
彼らはそれを市場で販売するかもしれません。  
 大きなソフトウェアおよびサービス会社のほとんどは、現在、数千から数十万ドルの範囲のバグ報奨金を提供しています。もう1つの極端な例は、サイバー武装メーカー（軍事および諜報機関に販売する）や法医学会社（法執行機関に販売する）などの前略者に販売するためにエクスプロイトを購入する事業者です。  
研究者は、バグをベンダーに直接開示することもできます。現在、多くのベンダーは、少なくとも市場規模と一致するように市場価格を一致させようとする、開示された脆弱性に対して報酬を支払うバグ報奨金プログラムを用意しています。  
 たとえばAppleは、ユーザーによるクリックを必要とせずにiOSカーネルをハッキングできるすべての人に100万ドルを提供しています。  
 大きなバグの恩恵の欠点は、以前はバグが自然に発生していたものの、たとえば、コードが最終的に重要なプラットフォームになってしまうオープンソースプロジェクトへの寄稿者によって意図的に導入されていることです。  
ベンダーがパッチを発行する前にエクスプロイトが実際に使用されている場合、それはゼロデイと呼ばれ、通常は標的型攻撃に使用されます。  
 システムへのパッチ適用に失敗した顧客は、犯罪組織が大規模に展開できる複数のエクスプロイトに対して脆弱になります。  
1。  
 コストを最小限に抑える必要がある場合はパッチを適用します。コードに多数の製品バージョンが含まれている場合、多くのテストが含まれる場合があります。  
2。  
 怠惰な顧客はパッチに失敗し、結果として感染する可能性があります。  
）3。  
4。  
セキュリティエンジニアリング895ロスアンダーソン27。  
 方法論5。  
6。  
 優れたデバイスには、Microsoftの火曜日のパッチなどの定期的なイベントに対処する自動化機能が組み込まれており、数百万ものIoTデバイスの更新やリスク評価は、今後数年間頭痛の種となるでしょう。  
1990年代の議論は、製品のパッチを数か月または数年もパッチを当てないままにしておくことに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引き起こされました。  
 これにより、調査担当者がベンダーに提供する必要のあるスペースについて、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]。  
 最終的な開示の脅威により、ベンダーは彼らの尻を手に入れました。遅延により、修正をリリースする前に適切にテストするのに十分な時間が与えられました。研究者は自分の履歴書を作成するための信用を得ました。顧客はバグレポートと同時にバグ修正を受け取りました。そして、大企業は、彼らの法人顧客が計画できる定期的な更新を組織しました。  
これは、セクション26で説明した契約の一部でした。  
7。  
27。  
7。  
 これはMitreによって維持されますが、CVEの割り当てを大規模なベンダーに委任します。  
 脆弱性の重大度の数値表現を提供するCommon Vulnerability Scoring System（CVSS）があります。  
5EUはこれらのCSIRTの名前を変更しています–コンピュータセキュリティインシデント対応チーム。  
5。  
S  
 これらのリソースは、脆弱性と更新の追跡を自動化するために重要です。  
システムが機能し始めたとき、それはセキュリティ経済学者による研究の主題になりました。  
 プラグマティストは、理論的および経験的観点の両方から、ベンダーにパッチを適用させるには開示の脅威が必要であると主張しました。  
6。  
 それ以来、大衆市場ユーザー向けの自動アップグレードの導入、脆弱性のある市場を作る企業の設立、そしてバグがどの程度関連しているかに関する実証的研究を行ってきました。  
27。  
7。  
 セクション4。  
1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、オンラインで利用可能な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ーシステムの脆弱性を発見し、責任を持って開示した後、どのように訴訟を起こし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、バグバウンティプログラムや毎月のパッチワークを実行すると、人々を訴えるよりも優れていることを知るまでに20年以上かかりましたが、レガシー業界の多くの企業は、製品にさらに多くのソフトウェアが含まれているにもかかわらず、まだうまくいっていません。  
 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道を譲りました。  
 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Arm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するとどうなりますか？これは、LinuxのShellshockバグやOpenSSL（影響を受けるLinux）のHeartbleedバグのように、外傷的なものになる可能性があります。  
5。  
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を組み込んだ何千もの製品があります。  
 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。  
このようなショックへの対処は、2010年後半にセキュリティインシデントとイベント管理という独自の専門分野になったプロセスの1つの側面にすぎません。  
5。  
3セキュリティインシデントおよびイベント管理脆弱性または攻撃について知ったときに行うことに対するインシデント対応計画が必要です。  
 今日、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）が学習するよりも遅くないようにしてください。  
 一部のアプリケーションでは、専門の企業から脅威インテリジェンスデータを取得できますが、IoTベンダーの場合は、自社のハニーポットを運用して、製品を攻撃している人々に即座に警告するのが賢明な場合があります。  
 次のアップグレードへのポイント、宝くじ、さらには現金など、いくつかのインセンティブを提供することも考えられます。  
5。  
次に、問題を修復できる必要があります。  
 今日では、脆弱性レポートから重大な違反まで、あらゆるものに対する調整された応答を準備することを意味します。  
4。  
3および脅威インテリジェンスチームを介して、責任のある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します。  
 衛星電話が必要かどうかを考えるのをやめたことはありますか？3番目に、パッチを迅速に展開できる必要があります。すべてのソフトウェアが独自のサーバーで実行される場合、それは簡単ですが、何百万ものコンシューマデバイスのコードにパッチを適用する必要がある場合事前の計画が必要です。  
5。  
 サプライチェーンが長くなるほど、関心のある競合の管理が難しくなります。  
最後に、信頼性を維持し、損害を制限するために、セキュリティ違反について迅速かつ正直に対処する必要性について、CEOとメインボードディレクターに事前に教育する必要があります。  
 したがって、緊急に必要になる可能性のあるすべての人の携帯電話番号と自宅電話番号を知る必要があります。  
 あなたが必要とする最後の事柄は、何十人ものジャーナリストが電話をかけて、広報担当者や交換機オペレーターさえも、バグを修正するために激しく奮闘しているときに妨害されることです。  
 これは、最初の（またはおそらく2番目の）ジャーナリストが電話をかけるとすぐに発送されます。  
 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遭遇し、銀行がそのすべてに対して訴訟を起こすと予想するかもしれません。  
 それが複数回発生した場合、顧客を失うことが予想されます。customerchurnは、通知された1回の違反後は2％に過ぎないかもしれませんが、2回の場合は30％、3回の場合はさらに30％です[2037]。  
27。  
8組織のリスクの不適切な管理組織の問題は、組織のメモリが失われ、変化する環境を監視するメカニズムがないため、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で触れた、組織内での人々の行動に関する大規模な文献があります。  
7、そして私はさまざまな章でさらに多くの例を挙げました。  
 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関与している場合、メンバーはランダムに互いに話すことができません。または、セキュリティエンジニアリング899ロスアンダーソン27。  
 方法論get chaos;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経由でルーティングすることはできません。  
 横方向の相互作用のないN人のクリーンな軍事指揮系統の通信の複雑さはlog Nです。誰もが他の人に相談する必要がある場合、それはN 2です。そして、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、それは2Nに向かうことができます。  
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。  
 私の学部の講義の例では、過度に野心的なプロジェクト、不十分な仕様、および実際のテストが行​​われていないため、ロンドンで1日間救急車のカバーがありませんでした。  
その日も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のシステムが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部分が遅れているかまったく行われないという従来の経験を受け入れると、多くの人々当然のことながら、1999年末にはかなりの数のシステムに障害が発生すると想定し、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への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かった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。  
 Y2Kbug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年以降も機能させ続けたい」。  
 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。  
5。  
 そのため、有能なチームは最初に困難な問題に取り組み、不確実性を減らします。それがDARPAの使命であり、スパイラルモデルの中核です。  
 しかし、正しく定義されたプロジェクトであっても、これを誤解するのは簡単です。  
2。  
リスク管理は、問題が制限のないセキュリティではさらに悪化する可能性があります。  
 1990年代後半、私たちは安全なスマートカードを手に入れようと考えました。その後、微分電力分析が行われました。  
 また、Apple製品はマルウェアに感染することができず、顔認識は本当のプライバシーの脅威になるには不十分だと考えていました。  
ミドルマネージャーは、チェックリストを下に向かってボックスでチェックすることで実装できるアプローチを好みますが、不確実性とオープンエンドのリスクに対処するには、アラート、詐欺、または注意に注意を払うオープンラーニングのプロセスが必要です。安全インシデント、または顧客の苦情-あなたが学ぶことができるものは何でも。  
 セクション9で書きました。  
6認定されたプロセスは、批判的思考に取って代わる強い傾向がありました。システムの保護要件を絶えず確認するのではなく、設計者はチェックリストにたどり着きます。  
 最初に困難な問題に取り組まないことで、不確実性が隠され、後でさらに悪化します6。  
 私たちのセキュリティの専門家は、問題が発生する可能性を予測するために賢明な人々にお金を払うように会社に働きかけることと、マネージャーを少なくして安価なエンジニアを使って製品をより早く提供することに、永遠の緊張があります。  
古典的な問題は、攻撃がインサイダーと外部のどちらから行われるかです。  
 しかし、政府機関や他の独占者でない限り、すべての状況を互いに、そして顧客にとって役に立たないように訓練することはできません。  
 たとえば、銀行の管理者は、デュアルコントロールの安全ロックによって家族が人質にされるリスクを軽減できることを知っています。大規模な取引で2つの署名が必要なことは、何か問題が発生した場合に負担がかかることを意味します。規制やその他の保護手段を文化的に埋め込むことは大変な作業です。複数の企業と接触した場合、コード監査（技術専攻が要求するもの）から追尾（半導体企業が防止するのに苦労している）から人々が次の章でISO 27001について説明します。  
セキュリティエンジニアリング901ロスアンダーソン27。  
 方法階段を上り下りするときの手すり（エネルギー会社のお気に入り）。  
2。  
3、しかしより広範なセキュリティ文化は大きく異なります-そして問題。  
 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であることは無責任な仕事です。  
 多くの組織では、昇進は年功序列と接触の問題です。したがって、CEOになりたい場合は、途中で他の人を止めずに20年かけて階層を登る必要があります。  
 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ダンプされるようにすることです。  
 これについてはセクション12で説明しました。  
2銀行が決済システムの詐欺責任をカード所有者、商人、またはその両方にシフトしようとする状況では。  
 もう1つの例はヘンリーフォードで、彼の車の1つで負傷した場合は、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状態の売上高は高く、契約状態に大きく依存しています。冗長性に対する恐れは、忠実な立場を不正な求職者に変える可能性があります。  
 したがって、大き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、問題が発生したときに後で誰が責任を取るかを自問してください。  
 確かに、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。  
6。  
2。  
 もちろん、これはめったに起こりません。  
6チームの管理安全で信頼できるコードを開発するには、適切な文化、適切なスキルの組み合わせ、適切なインセンティブを持つチームを構築する必要があります。  
 では、補完的なスキルを持つ強力な開発チームをどのように構築しますか？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、さまざまな作家が自分のスタイルや企業文化を反映しています。  
27。  
1エリートエンジニア1960年代にさかのぼるフレッドブルックスの有名な本「The Mythical Man-Month」は、世界初の大型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。  
 考えてみると、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り、管理職に昇進させて「失う」よりも、上級管理職の給与と尊敬をもって彼らのための投稿を作成します。  
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を取得し、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様の状況にあり、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り、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、国際的に知られた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科学者まで、複数のスーパースターを採用しています。  
 セキュリティエンジニアリング903ロスアンダーソン27。  
 チームを管理する彼らの生活をソフトウェアの作成に費やしてソフトウェアを上手に使いたいなら、敬意を払う必要があります。ただし、組織が給与、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フリであろうと、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、「科学者はトップに立つべきではなく、タップする必要があるということです。  
 そのうちの1つでは、ソフトウェアエンジニアは5年後にマネージャーになるか、ジュニアペイグレードのままでいることが期待されています。  
6。  
27。  
2Diversity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、GraceHopperやDame Stephanie Shirley（「SteveShirley」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。  
 少数派もありました。水星、ジェミニ、アポロのミッションの軌道計算は、アフリカ系アメリカ人の女性、キャサリンジョンソンが主導しました。  
 1990年代に私がアカデミーになったときから、地元のコンピューターサイエンスの学生の約6分の1が女性でしたが、女性の学生をさらに募集するというかなりの努力がありました。  
（南ヨーロッパと東ヨーロッパから多くの学生を受け入れることにより、性別のバランスを改善しました。  
 これは文化的な問題であり、それがどのようにして生じたのかについては多くの議論があります。  
5。  
コンピュータサイエンスの分野でも、多くのサブカルチャーのバリエーションが見られます。  
 セキュリティの分野では、暗号学とオペレーティングシステムの内部に重点が置かれた1990年代には圧倒的に男性でしたが、今でははるかにバランスが取れており、設計、ユーザビリティ、心理学の重要性を受け入れています。  
 女性の教員がいる研究グループは、有能な女性からより多くのアプリケーションを得る7。  
 それは3つ以上を意味するかもしれません。  
 明らかにいじめっ子を雇うことは悪い考えですが、事前に見つけるのは難しいかもしれません。  
セキュリティエンジニアリング904ロスアンダーソン27。  
 チームを管理して保持することは、仕事ではなく人を管理することのアイデアです。  
 いじめっこもしばしば忍び寄ります。彼らの下にいる人々を彼らの上にいる人々に吸い上げるのと同様に。  
 この管理スタイルが組織全体に広がっている場合、私のアドバイスは他の場所に行くことです。  
6。  
多様なスキルも必要です。  
 タスクによっては、ユーザビリティテストを主導するユーザー支持者がいる場合があります。安全またはセキュリティの擁護者;全体的な設計をクリーンかつ効率的に保つことを仕事とするアーキテクト。 APIを心配する言語弁護士、回帰テスト機械を実行するテストエンジニア、静的および動的分析ツールを保守するツールスミス。  
 各開発者に習得しようとするこれらのスキルの数、およびチームを超えて作業したり、コンサルタントに加わったりする対象分野の専門家の数を考慮する必要があります。  
 優れたテック企業は、会社全体でエンジニアをゆっくりと回転させて、企業にとっての価値を最大化するさまざまなスキルを獲得します（他の企業にとっても価値を最大化し、離れやすくします）[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回の正式な進行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でさえ）を静かに修正するのは悪い習慣です。バグは相互に関連しているため、さらに多くの可能性があります。  
 良い慣行の例として、航空管制では、エラーを起こしたコントローラーがそれを修正するだけでなく、オープンな抗議ですぐにそれを宣言することが期待されます。「私はSecurity Engineering905Ross AndersonでSpeedbird 123を持っています27。  
 誤ってターミナルコントロールエリアでチームのレベル8のゼロを管理し、ゼロを6つに下げるように指示しています。  
 ソフトウェアはそれほど劇的ではありませんが、問題はありません。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つのシグナルは、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何をしていたと思ったのか知りたいです：itaの設計エラー、それともコーディングの失敗ですか？しかし、チームは「適切な」量とコメントのスタイルについて簡単に戦うことができます。後でコードを読んでバグを理解するのに十分であれば、スタイルが何であるかはそれほど重要ではありません。しかし、人々が受け入れる一貫したスタイルがあり、それが目的に適合していることが重要です。  
27。  
4創発的なプロパティ1つの議論は、自分のコードをセキュリティで保護する責任を全員に負わせるか、それとも誰もが信頼するセキュリティの第一人者になるかどうかです。  
 大手企業が発見した答えは「両方」です。  
 MicrosoftとGoogleはどちらも、セキュリティの「ブートキャンプ」を通じて新人エンジニアを配置しているため、誰もが基礎知識を理解し、さまざまなレベルで主題の専門家を抱えています。  
秘訣は、チームの専門分野の量を管理することと、スペシャリスト（セキュリティアーキテクトやテストの第一人者など）が他の開発者と対話する方法にあります。  
6。  
 プロの開発チームは、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。  
セキュリティエンジニアリング906ロスアンダーソン27。  
 チームの管理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生成器などの暗号の問題を回避できます。  
 たとえばCoverityの場合、著者は、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20,000を超えるバグレポートが突然トリアージされ、出荷日が数か月[235]。  
 暗号化ライブラリの場合は、第6章で、ECBなどの弱い動作モードがデフォルトとして使用される傾向があるため、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と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、時間を節約するだけでなく、従業員に力を与えるのに役立ちます。  
 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、プログラミング言語に依存して、タイプセーフティメカニズムを使用してAPIの問題を回避します。  
セクション20で見ました。  
 100を超えるさまざまな電話モデルで使用されているモバイル電話のベースバンドソフトウェアから、100を超えるさまざまな車で使用されている車両コンポーネントまで、他の多くのアプリケーション領域にも同様の緊張があります。  
27。  
6そして最後に。  
。  
技術管理はジェネラリストが行うことはできません。8人事の主な仕事は被害の制限であり、退職者があなたを訴えることを防ぎます。  
7。  
 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。  
 インテックを成功させるマネージャーは、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である場合があります。  
 彼らはあなたのエンジニアリング状態に道徳的なリーダーシップを提供し、顧客に安心感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、あなたの会社には1つ以上の技術職がいるということです。  
7要約セキュリティ、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。  
 つまり、他の機能、および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と共に、ソフトウェアのセキュリティが必要です。  
 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解明することです。  
 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支援、そして何よりもセキュリティの向上に取り組む環境の構築には、多くの課題があります。能力。  
”セキュリティエンジニアリング908ロスアンダーソン27。  
 概要研究の問題この章で説明する問題は、私たちの分野で最も困難で最も重要なものの1つです。  
 これらの各インターフェースは、研究の生産的な領域になる可能性があります。次の論理的なステップの1つは、知っていることを安全経済学および安全ユーザビリティと統合することです。  
 すべての経験豊富な開発者またはセキュリティコンサルタントは、悪意のあるインセンティブ、有毒な文化、高い離職率、無能な管理、およびディルバートの漫画で見られるその他すべての事柄について、企業に関するホラーストーリーを共有しています。  
 きちんとした経験的基盤を与えられたものは何でしょうか？故ジャック・ヒルスレイファーは、マネージャーが過ちから学ばざるを得ない組織を設計しようとするべきだとの見方をしていました。脅威環境の変化を監視し、システム開発だけでなく、内部統制などのサポート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しますか？コードとしての管理のようなものが必要なのでしょうか？従業員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？そして、組織の文化人類学は何を言わなければならないでしょうか？前の章では、アルカイダによってもたらされたと思われる斬新な脅威に対する政府の対応がさまざまな点で不適切であったことを確認しました。社会的レジリエンスの予算の多くは、テロ対策劇場に費やされ、他の社会的リスクへの備えが犠牲になったパンデミックとして。  
 その結果、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、セキュリティ開発の取り組みの多くがコンプライアンスを目的としています。  
 FredBrooksの「Mythical Man Month」[328]やNancy Levesonの「Safeware」[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。  
7。  
 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、自動車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、これは執筆時点ではまだ草案にすぎません。  
1。  
 Henry Petroskiは、橋梁建設の歴史、橋梁が落下する理由、および土木技師が崩壊からどのように学んだかについて話し合います：起こりがちなのは、確立された設計パラダイムが拡張され、予期せぬ理由で突然失敗するまで拡張されることです。 1518]。  
 セキュリティ障害については、Schneier onSecurity、Krebs on Security、SANSなどの主要なセキュリティブログ、業界紙をフォローすることが重要です。  
 多くのビジネスアカデミーは空港の本を販売するのに適したビジネスを賞賛していますが、私たちが必要とするのは、組織がどのように失敗するかについてのより重要な理解です。  
 Charles Handyなどの組織理論家は、権力、役割、タスク、人、またはそれらの組み合わせに基づいた文化を持つ企業について語ります。  
 おそらく、そのような洞察は、人々が実際に最もよく働く方法をサポートするより効果的なツールとワークフローを設計するのに役立つかもしれません。