Section 2: Week 4: Design a Control Model for Secure Development

Nate Bachmeier

TIM-7030: Managing Risk, Security, and Privacy

June 28th, 2020

North Central University

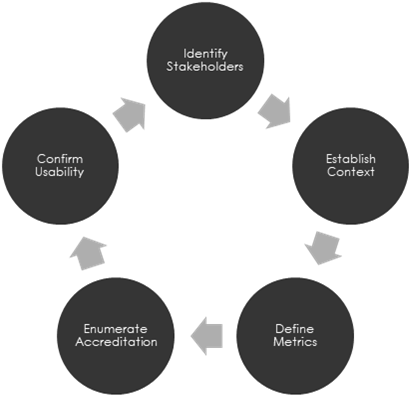
# Design a Control Model for Secure Development

NCU Financial (NCU-F) is a matrix organization that uses decentralized development teams to build FinTech solutions. This software approach enables the business to release features rapidly but at the cost of inconsistent processes and maturity levels. Instead, the senior leadership team wants a model that promotes best practices around controls and secure software design. An upcoming project replaces the Enterprise Resource Planning (ERP) system, enabling customers to self-service various financial transactions. This ERP system will also serve as the example basis for that development model and used across future deliverables.

# Section I: Building the Model

Building a secure software development process requires a feedback loop that identifies stakeholders, establishing priorities, audits accreditation requirements, and releases a usable product (see Figure 1). In parallel to this process are frequent requirement gatherings and considerations for data and application controls.

Figure 1: Building a model



## Identify Stakeholder Incentives

Implementing enterprise software solutions requires teams of professionals from across multiple disciplines, each with a unique perspective. Alignment and inclusion of these different perspectives are critical, or it causes distortion, leading to political confrontation and critical misses. These diverse ideas originate from industry norms that seek to improve consistency, accuracy, and efficiencies for different business aspects (see Table 1). While there are advantages for teams to operate in vertical silos, such as the speed of decisions, an integrated framework draws on the expertise across the company (Nicho, Khan, & Rahman, 2017). When all stakeholders can participate in the process, it culminates in releasing the right product at the right time.

Table 1: Stakeholders

|  |  |
| --- | --- |
| Team/Role | Primary Concerns |
| Customers and other end-user | Privacy, reliability, usability, and available |
| Security Engineers | Confidentiality, integrity, and availability |
| Software Engineers | Reliability, observability, and performance |
| Program Managers | Functionality, predictability, and extensibility |
| Support Engineers | Observability and supportability |
| User Experience (UX) | Convenience and performance |
| Legal and Compliance | Privacy and confidentiality |
| Marketing and Sales | Functionality and consistency |
| Accounting and Finance | Economical and profitable |
| Senior Leadership | Connects with the broader company strategy |
| Technical Writers | Consistency and explainability |
| Network Security Engineers | Authentication, authorization, and auditing |
| Operations | Observability, reportability, and discoverability |

## Identify Enterprise Context

The stakeholders need to reach a consensus on the overarching strategy for implementing the ERP system. This strategy must be cognizant of the enterprise context (see Table 2) as part of appropriate scoping and costing decisions (Rafeq, 2019). For instance, building a custom solution from scratch requires more resources than NCU-F has available and forces the business to consider acquiring and extending third-party software. Next, the organization must split the custom development work into buckets for in-house versus outsourced teams. While in-house sourcing provides the most oversight and control, it can detract from other commitments and core competencies. For example, the ERP must schedule the execution of a sophisticated credit processing framework. Building the job scheduling components is a generic problem and more easily outsourced. It frees up NCU-F engineers to focus on the value differentiating aspects of the feature.

Table 2: Enterprise Context

|  |  |
| --- | --- |
| Aspect | Questions |
| Company Mission | Why does the company exist |
| Company Strategy | Which markets and approaches are acceptable |
| Risk profile | How aggressive or conservative is appropriate |
| Existing commitments | What other deliverables are competing for resources |
| Industry-specific challenges | Which compliance, regulatory, and competitive analysis needs consideration |
| Resource Availability | Does the business have sufficient capital and human capacity to execute the plan |
| Core Competencies | What talents and expertise does NCU-F possess |

## Establish Timelines and Priorities

Inadequate planning, scope creep, and poor communication are three common reasons that projects fail (Jain, 2018). Partial mitigation comes from an iterative design that articulates the barebones solution and lays out a path through multi-release strategies. For instance, the ERP system has rigid requirements for handling mortgages, individual retirement accounts, and private equity investments. Not enough time exists to deliver all three scenarios, so the company must address the schedule risk through acceptance, transference, or avoidance strategies (Baskerville, Rowe, & Wolff, 2018). Choosing between these options requires assessing the inherent risks before deciding which outcome is most palatable (see Table 3).

Table 3: Schedule Risk

|  |  |  |
| --- | --- | --- |
| Strategy | Examples | Inherent Risks |
| Acceptance | Extend the deadline Apply additional pressure to teams | Nothing can release Missed sales opportunities Increased team attrition rates |
| Transference | Shift resources from other projects | Context switching is expensive Other commitments missed Outside assistance adds drag |
| Avoidance | Cut the new feature Parallelly run legacy/new systems | Customer expectations missed Increased operational costs |

## Performing Database and Application Testing

The credit card processing module takes one minute to complete and succeeds 99% of the time. Is that outcome good or bad? Answering this question in a vacuum is impossible. It requires an agreed-upon Service Level Objective (SLO) and accompanying Quality of Service measurements (QoS). QoS models measure a scenario in terms of reliability, availability, response time, and throughput (Jammal, 2017). If these measurements are highly variable, it indicates controls are missing from the system (see Table 4). For instance, when a customer uploads their mortgage application, they expect to promptly perform that action before the system responds with a confirmation number promptly. However, if that application becomes lost due to an internal service rebooting, then artificial processing delays are introduced. These classes of issues increase support costs and reduce the credibility of NCU-F. While impossible to discover every defect in advance, controls such as service replicas and message durability can improve the likelihood of presenting a positive experience.

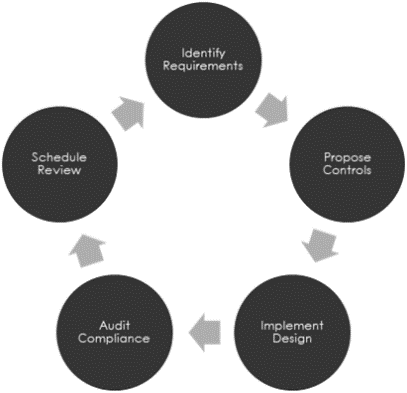
Table 4: Measurements of QoS

|  |  |  |
| --- | --- | --- |
| Measurement | Description | Example Controls |
| Reliability | The likelihood of a valid transaction succeeds | Event durability and retry policies |
| Availability | An endpoint’s ability to respond to service requests | Load balancers and eventual consistency models |
| Response Time | Time to complete the transaction | Quota Management |
| Throughput | Number of supportable parallel transactions | Elastic and virtual compute solutions |

## Completeness, Compliance, and Accreditation

Businesses within regulated environments can only transact with vendors that meet specific compliance and accreditation standards. These regulations come from industry, state, and national levels, like health care’s Health Insurance Portability and Accountability Act (HIPPA) or Europe’s General Data Protection Regulations (GDPR). Navigating these legal frameworks is highly-complex and requires experts to provide guidance and review decisions. While implementing these controls, a Plan-Do-Act-Check (PDAC) feedback loop can be appropriate for ensuring the controls are sufficient (see Figure 1) (Radhakrishnan, 2015). For instance, GDPR states that businesses must secure user privacy or pay severe fines (Kovacs, 2017). Perhaps the initial release has sufficient controls, but future features might increase the attack surface and cause the system to fall out of compliance. Alternatively, the European Union might change their definition of user privacy or include additional control requirements (e.g., auditing). Detecting and preventing these policy drifts necessities periodic auditing and compliance assessments.

Figure 1: Accreditation Loop



## Usability Testing

The system’s audience can include people with international addresses, Unicode characters in their names, and similar locale-specific data entry decisions. Addressing these challenges requires the stakeholders to identify which markets will consume the product and what cultural norms might exist (see Table 5). For instance, if the ERP system must service customers with slow and unreliable networks, then specific design decisions like limiting the inclusion of high-resolution graphics, audio files, and streaming video might be necessary. When these bandwidth considerations are lacking, the product is virtually unusable and unavailable, preventing that market from being served.

Table 5: Usability Challenges

|  |  |  |
| --- | --- | --- |
| Aspect | Description | Challenges |
| Internationalization (I18N) | Supportability of foreign standards | Text encoding  Validation Patterns |
| Localization (L10N) | Support for different languages | Date time formats Text widths |
| Device Specifications | Norms at the physical layer | Slow internet connections Small screens |
| Tediousness | Are everyday tasks trivial | Varying technical user levels  Guessing over measuring |

## Software Development Lifecycle (SDLC)

SDLC includes planning, analysis, design, implementation, maintenance, and termination stages (Wikipedia, 2020). Discovering and resolving issues earlier in this development lifecycle are substantially cheaper than later on. For instance, correcting an issue during the planning phase might entail updating a Microsoft Visio diagram. Meanwhile, changes in the maintenance cycle could require multi-release updates to correct production environments without encountering downtime. Introducing controls and processes between each step will improve a successful project's chances (see Table 6). For example, changing the scope during the implementation phase or lacking executive sponsorship early on will jeopardize a timely release.

Table 6: SDLC

|  |  |  |
| --- | --- | --- |
| Phase | Description | Controls |
| Planning | Abstract proposal of feature | Procedural approval processes Executive sponsorship |
| Analysis | Investigation into feasibility | Forecast market interest Forecast acceptable budget |
| Design | Formalization of the plan | Agreement on scope, timelines, and iterative release strategy Acceptable QoS/SLO metrics |
| Implementation | Execution of the plan | Periodic checkpoints Stakeholder audits |
| Maintenance | Available into production | Maintenance and Release windows Authorized deployment team |
| Termination | Decommission and replacement | Archival procedures Data migration validation |

# Section II: Recommendation for data and application controls

The primary objective of data controls is to ensure information confidentiality, integrity, and availability against negligence and malicious threats. Data management systems enforce these expectations through encryption, digital signatures, and backups, among other strategies (see Figure 2). While many protections exist, incorrect application controls can provide attack vectors to bypass these systems. For example, Alice cannot directly access a production secret but can delegate the request through a web server (see Figure 3). In this situation, authentication, authorization, and auditing (AAA) controls are missing. These elevation paths can be challenging to uncover without tooling and formal threat modeling (Hagestad & Straumann, 2017). Perhaps Alice can authenticate and control the management service and then fetch the secret through an intermediary. Similar to the previous example, the mitigation requires passing the caller identity so that AAA checks thoroughly protects the path. Under specific scenarios, adding these checks is impossible, and that requires more reliance on auditing and detection policies.

|  |  |
| --- | --- |
| Figure 2: Control Taxonomy | Figure 3: Application Controls |

# Section III: Comparison of development strategies

There are several development process strategies available, each with its strengths and weaknesses (see Table 7). While many enterprises see Waterfall as an anarchic relic and fully embrace Agile, this perspective neglects to account for the organization’s culture, timelines, and objectives. For instance, software that is difficult to patch, like IoT devices, is likely to gain upfront planning strengths. There can also be value in using different strategies during specific stages of the project. For example, during the more ambiguous beginning, using RAD to prototype to iterate quickly. Once the design formalizes, transitioning to a Spiral or similar can follow a ‘Plan-Do-Act-Check’ loop that addresses the most critical items first.

However, agile methodology is fundamentally a greedy algorithm, which continuously seeks the local optima by performing the task that gains the maximum immediate benefit. In contrast, frameworks like Waterfall focus on longer-term planning for reaching the global maxima. A potential optimization exists where Waterfall-like strategies continue to occur within the leadership team versus agile remains at the lower-levels. Through a combination of long and short-sighted decisions, the organization is both directed and adaptive to change.

Table 7: Development Strategies

|  |  |  |  |
| --- | --- | --- | --- |
| Process Type | Approach | Strengths | Weaknesses |
| Waterfall (CodeAcademy, 2019) | A linear sequence of well-defined stages | Focus on early stages of requirements gathering | Clients might lack understanding of what they want  Assertions disproven later are costly |
| Spiral (Easy Learning, 2020) | Loop of planning, risk assessment, development, and evaluation | Focus on what is the next immediate requirement | Large projects require many iterations, and each loop adds costs |
| Rapid Development (Idesis, 2017) | Software is clay, not steel.  Rapid prototyping with continuous customer feedback | Deliver sooner, with frequent iterations  Removes unnecessary work and process  Ideal for internal tools | No focus on team structures or communication  Hard to scale across large teams, meeting heavy |
| Reuse (Time2Pretend54, 2017) | Find reusable libraries and open-source project, then build from there | Quickly build solutions by gluing together components  It saves time and money | Less customizable and inflexible  Potential licensing challenges |
| Extreme Programming (Udacity, 2015) | Lightweight methodology for small to medium teams with vague requirements | Mentality of sufficiency  Frequent releases with numerous checkpoints  Defer building features until needed | An idealistic model that is impractical for many businesses  Challenging to use outside of internal infra teams (Parogi, 2016) |

# References

Baskerville, R., Rowe, F., & Wolff, F. (2018). Integration of information systems and cybersecurity countermeasures. *SIGMIS Database. Volume 49*, 33-52. doi:https://doi.org/10.1145/3184444.3184448.

CodeAcademy. (2019, February 22nd). *What is Waterfall?*. Retrieved from YouTube: https://youtu.be/LxEmGNgqYJA

Easy Learning. (2020, April 4th). *LEC-06 Spiral Development Model*. Retrieved from YouTube: https://youtu.be/fTfoB1h2km4

Hagestad, B., & Straumann, A. (2017). Commentary: Collaborating to Achieve a Mutual Cybersecurity Advantage. *Biomedical Instrumentation & Technology, 51*(1), 34-39. doi:https://doi.org/10.2345/0899-8205-51.1.34

Idesis, S. (2017, October 17th). *What is rapid application development*? Retrieved from YouTube: https://youtu.be/JHcxbGwHtsY

Jain, N. (2018, February 21st). *Top 10 reasons for project failure*. Retrieved 28 2020, June, from Whiz Labs: https://www.whizlabs.com/blog/top-10-reasons-for-project-failure/

Jammal, M. &. (2017). Software-Defined Networking: State of the Art and Research Challenges. *Computer Networks, 72*. doi:https://10.1016/j.comnet.2014.07.004

Kovacs, L. (2017). Cybersecurity policy and strategy in the European Union and NATO. *Revista Academiei Fortelor Terestre, 23*(1), 16-24. Retrieved from https://search-ebscohost-com.proxy1.ncu.edu/login.aspx?direct=true&db=a9h&AN=128789540&site=eds-live

Nicho, M., Khan, S., & Rahman, M. (2017). Managing Information Security Risk Using Integrated Governance Risk and Compliance. *International Conference on Computer and Applications* (pp. 56-66). Doha, United Arab Emirates, United Arab Emirates. doi:https://doi-org.proxy1.ncu.edu/10.1109/COMAPP.2017.8079741

Parogi, J. (2016, June 10th). *Five reasons why eXtreme programming is not popular*. Retrieved from Medium: https://medium.com/agility-path/5-reasons-why-extreme-programming-isnt-popular-83790418b901

Radhakrishnan, S. (2015). COBIT Helps Organizations Meet Performance and Compliance Requirements. *COBIT Focus, 5*, 1-5. Retrieved from https://search-ebscohost-com.proxy1.ncu.edu/login.aspx?direct=true&db=bth&AN=102026122&site=eds-live

Rafeq, A. (2019). COBIT design factors: a dynamic approach to tailoring governance in the era of digital disruption. *COBIT Focus*, 1-6. Retrieved from https://eds-b-ebscohost-com.proxy1.ncu.edu/eds/pdfviewer/pdfviewer?vid=1&sid=4f3fc652-db5c-4e6e-968a-ab8809af0d76%40sessionmgr103

Time2Pretend54. (2017, February 8th). *What is reuse-oriented software engineering*? Retrieved from YouTube: https://youtu.be/Dfi4cxCC\_I0

Udacity. (2015, February 23rd). *Extreme Programming (XP) - Georgia Tech - Software Development Process*. Retrieved from YouTube: https://youtu.be/hbFOwqYIOcU

Wikipedia. (2020, June 28th). *System Development Life Cycle*. Retrieved June 28th, 2020, from Wikipedia: https://en.wikipedia.org/wiki/Systems\_development\_life\_cycle