Strategic Prototyping for Developing Big Data Systems

Hong-Mei Chen, *University of Hawaii at Manoa*
Rick Kazman, *University of Hawaii at Manoa & Software Engineering Institute*
Serge Haziyev, *Softserve Inc.*

Abstract. Conventional evolutionary prototyping for small data system development is inadequate and too expensive for identifying, analyzing and mitigating risks in big data system development. This article presents RASP (Risk-based, Architecture-centric Strategic Prototyping)—a model for cost-effective, systematic risk management—and shows how this is deployed in agile big data system development. The RASP model advocates using prototyping strategically and only in areas that architecture analysis cannot sufficiently address. In RASP, less costly MVP (minimum viable product), throw-away and vertical evolutionary prototypes are used strategically, instead of blindly building full-scale prototypes. The RASP model is validated in an embedded multiple case study of nine big data projects with a global outsourcing firm. A decision flowchart and guidelines distilled from lessons learned for whether, when, and how to do strategic prototyping are provided.

Key words: Risk Management, Rapid Prototyping, Software Architecture

Introduction

Big data system development faces more and greater risks than traditional "small" data system development. These risks stem from the 5V (Volume, Variety, Velocity, Veracity, Value) characteristics of big data, paradigm shifts, rapid proliferation of big data technology, rapid technology changes, the difficulty in big data technology selection, complex integration of new and old systems, and the short history of big data system development [9][11]. For example, in small data systems, the focus of design is on data at rest (i.e. stored in the database) whereas in big data systems there is far more emphasis on data in use and data in motion, which drives data management to an unfamiliar territory such as NoSQL (Not Only SQL). Moreover, in "small" data system development data-program independence was a core development principle. Architecture decisions were relatively trivial. In big data system development data-program *dependence* is the norm, and thus architecture design becomes critical and poses great risks if done poorly [11]. Many new architectural patterns have recently become dominant in big data development, such as horizontal scaling, data lake, polyglot persistence, Lambda architecture, and massively parallel processing. And the system development paradigm has shifted: from the primary effort focused on coding in small data system development to "orchestration" and technology selection in big data systems. There are a large number of emerging big data technologies (both open source and proprietary), many of which are provided by new vendors or open source projects; hence, risks in technology selection are heightened.

As such, all the cost, schedule and quality risks in traditional system development are more pronounced in big data system development. Managing—identifying, understanding and mitigating—cost, schedule, and quality risks is of course critical for system success, and the methods for managing risks differ in big data system development. Traditionally, small data management, following the RAD (Rapid Application Development) methodology, *primarily* and *implicitly* relies on horizontal evolutionary prototyping to help manage risks. We distinguish two types of evolutionary prototypes: (1) horizontal: all system functions are developed with limited, but increasing functionality per release. (2) vertical: one or more system components are developed with full functionality in each release. Evolutionary prototyping is needed when requirements are uncertain, technologies are new, no comparable system has been previously developed, and in cases where experimentation or design evaluation are required to assess solutions.

However, the evolutionary prototyping approach is challenged by the data *scale* and complexity of big data systems. For example, in our early studies, we observed that data retrieval performance with a Hadoop system of 100 nodes is dramatically different than results with a 500 node system—assuming a linear scale-up is naïve (and incorrect). The scarcity of benchmarks in big data systems further aggravate these problems. Creating prototypes has always been expensive and time-consuming, and more so in big data system development. And many risks cannot be identified by prototyping, such as (1) vendor or product survivability for commercial products, and (2) size and vigor of the community for open source products.

There are alternatives to prototyping such as architecture analysis, simulations and formal analytical models. Simulation is primarily viable in mature domains, where critical system properties may be abstracted and modeled. Formal analytical models are typically used in domains with critical requirements, such as hard real-time deadlines or safety-critical properties. But the cost, steep learning curve, and difficulty of creating and scaling such models makes them prohibitively expensive for most commercial projects. Architecture analysis has been practiced as a relatively inexpensive means of gaining early insight into the properties and risks of a system. For traditional small data systems architecture design is straightforward—it is an N-tier client-server—hence architecture analysis has been largely taken for granted and omitted. However, in big data systems, many paradigms shifts underscore the importance of an explicit architecture design and analysis process [11]. Architecture analysis—understanding the consequences of design decisions—is critical and should be performed early in the lifecycle. Architecture analysis not only identifies architecture-related risks (e.g., performance, availability, security, modifiability, integrability), but also process risks (e.g., requirements, tools, methods) and organizational risks (e.g., big picture, awareness, scope and unrecognized needs) [2].

Architecture analysis usually takes much less time and costs much less than prototyping. However, architecture analysis cannot address all risks; only prototyping can, for instance, verify real-time performance of a new big data component or assess runtime resource consumption thresholds, scalability limits, etc. Prototyping and architecture analysis each have pros and cons. Our research therefore centers on how to manage risks in agile big data system development utilizing both prototyping and architecture analysis *cost-effectively*. We consider two types of prototypes: throw-away and vertical evolutionary. Horizontal evolutionary is more typical of small data system development while for big data system development, vertical evolutionary prototypes are generally more suitable, as our case studies will show. We also consider a special kind of evolutionary prototype, MVP (Minimum Viable Product). A MVP has *only* those core features that allow the product to be deployed, minimizing the time spent on an iteration. A MVP emphasizes hypothesis testing by fielding the product with real users and collecting usage data. The data helps to confirm or reject the hypothesis, allowing additional design decisions to be made.

The cost (money, time and effort) and effectiveness issues drove us to advocate the use of prototyping *strategically* and *only* in areas that architecture analysis cannot adequately address, resulting in our creation of the RASP (Risk-based Architecture-centric Strategic Prototyping) model. By "strategic" prototyping, we refer to employing an architecture-based approach for first identifying and analyzing risks and then using less costly throw-away prototypes and vertical prototypes when necessary, in contrast to building full evolutionary (horizontal) prototypes blindly, as in conventional small data system development.

Empirical Basis for RASP: Nine Case Studies

The RASP model was developed based on the Collaborative Practice Research results of nine case studies developed by a global outsourcing firm, Softserve Inc. (hereafter SSV) to provide guidelines on employing strategic prototyping for risk management in the context of agile big data system development. The RASP model can stand alone but we have integrated it into our architecture-centric agile development methodology, BDD (Big Data Design) [9][11]. Our RASP model grew out of, and has been extensively validated via, these case studies. They cover a broad range of domains, including:

1. Network security, intrusion prevention
2. Online coupon web analytics
3. Cloud-based mobile app development platform
4. Telecom e-tailing platform
5. Web analytics and marketing optimization
6. Healthcare insurance operational intelligence
7. Ultra-large scale travel site
8. Operations intelligence platform for content delivery network
9. Big Data as a Service (BDaaS) platform

The nine studies that form the primary empirical basis for this research are described in detail in a companion web-site (http://itm-vm.shidler.hawaii.edu/bus311/BDCases/CaseStudies.htm). These studies, and the extensive practical experience of SSV, show that development of big data systems is qualitatively different from traditional small data system development. The majority of the schedule and effort in these studies went into value discovery (from big data), architecture analysis, prototyping, and orchestration of off-the-shelf commercial and open source components. In contrast, in small data system development a far higher percentage of effort and schedule goes straightly into programming. Given that the development process is different [9][11], we need different methods for designing and analyzing such systems. And given that these systems are large, complex, and risky, an important aspect of the new development process is determining whether, when, and where to prototype, as we will show.

An Architecture-Centric Approach

The RASP model is integrated with the BDD method [9][11] which is an architecture-based design method aimed at addressing the complexity inherent in big data (size, scale, heterogeneity, ambiguity, etc.). As Simon explained, architecture is the key to dealing with complexity: “The near-decomposability property of complex systems simplifies the behaviors and description (process) of a complex system, and makes it easier to understand” [15]. As big data paradigms have shifted the focus of effort from primarily coding to primarily orchestration, an architectural approach for technology selection and system integration is critical.

The relationship between architecture design and agility has been the subject of debate over the past decade. An architecture-centric approach facilitates rapid iterations with less disruption, as well as lower risk and cost, by creating appropriate architecture abstractions. For example, if it was anticipated that a technology family would rapidly change (with technologies introducing new capabilities or with new entrants to the technology family) then a prudent architect would insert an abstraction layer as an intermediary between the technology family and the rest of the system, to insulate the system from such changes. Hence an architecture-based mindset can lay the foundation to support agility in both initial system construction and future iterations in big data system development.

The BDD process on the left side of Figure 1, however, differs from typical small data system development processes [12]. Small data system development, based on data-program independence, typically consisted of 7 phases: requirement analysis, conceptual design, selection of DBMS, logical design, physical design, prototyping, implementation, and performance evaluation. In a multiple case study of 25 large enterprises, we discovered that big data is viewed as “high risk” and “disruptive.” Consequently, all 25 enterprises we interviewed started with a Value Discovery (VD) process [10]. This phase encompasses innovation processes, use case development, and strategic deployment planning, including cost-benefit analysis, sourcing decisions and talent management. The innovation process can be top-down, bottom-up or a mix. It often is a long process, taking up to 3 years for some companies. High level use cases that demonstrate the value of big data and opportunities for innovation are solicited or developed *prior to* any detailed requirements analysis activity. If companies do not find use cases that demonstrate value from big data, they typically do not continue with experimentation and deployment planning.

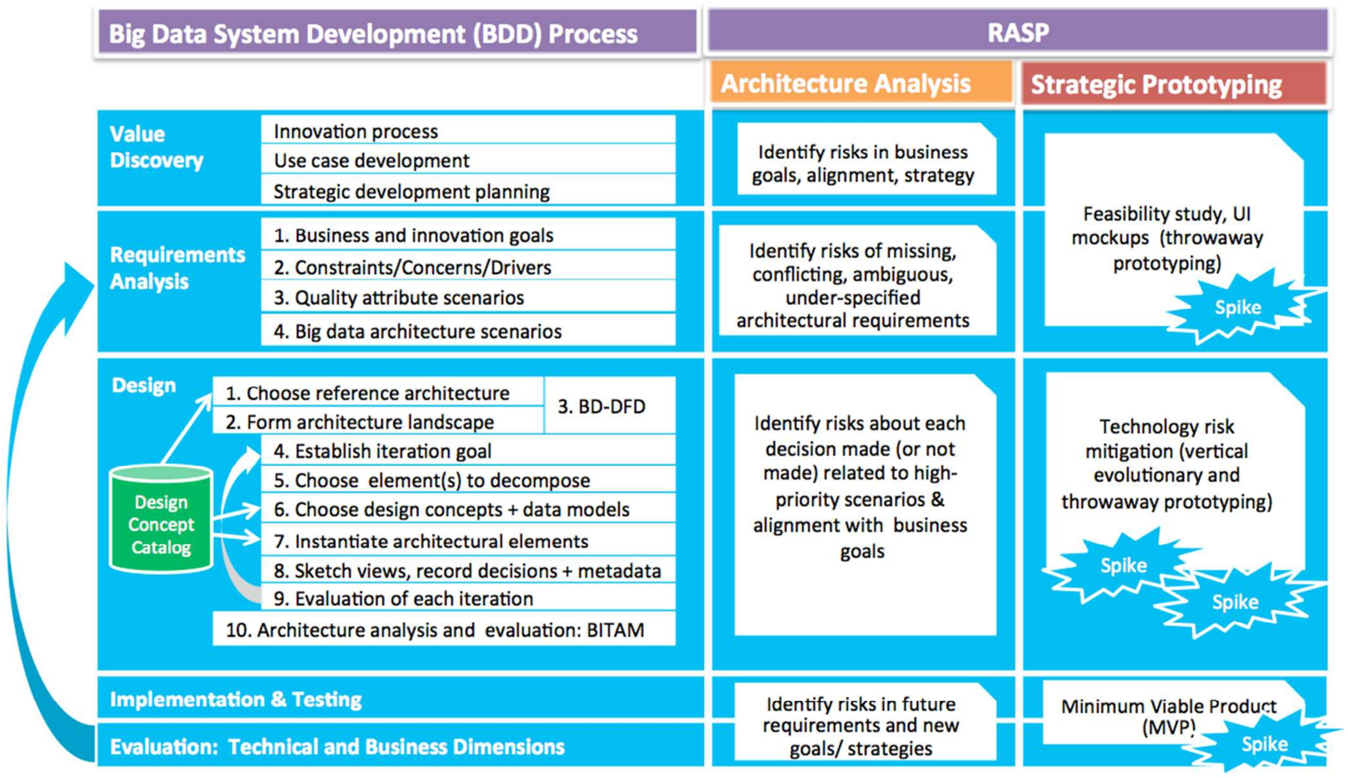

Big Data System Development (BDD) Process
RASP
Architecture Analysis
Strategic Prototyping
Value Discovery
Innovation process
Use case development
Strategic development planning
Identify risks in business goals, alignment, strategy
Feasibility study, UI mockups (throwaway prototyping)
Spike
Requirements Analysis
1. Business and innovation goals
2. Constraints/Concerns/Drivers
3. Quality attribute scenarios
4. Big data architecture scenarios
Identify risks of missing, conflicting, ambiguous, under-specified architectural requirements
Design
1. Choose reference architecture
2. Form architecture landscape
3. BD-DFD
4. Establish iteration goal
5. Choose element(s) to decompose
6. Choose design concepts + data models
7. Instantiate architectural elements
8. Sketch views, record decisions + metadata
9. Evaluation of each iteration
10. Architecture analysis and evaluation: BITAM
Design Concept Catalog
Identify risks about each decision made (or not made) related to high-priority scenarios & alignment with business goals
Technology risk mitigation (vertical evolutionary and throwaway prototyping)
Spike
Spike
Implementation & Testing
Evaluation: Technical and Business Dimensions
Identify risks in future requirements and new goals/ strategies
Minimum Viable Product (MVP)
Spike

Figure 1. The RASP Model Integrated with BDD

In the VD phase, architecture analysis based on a rough, early design can help estimate total cost of ownership and identify risks. This architecture analysis includes eliciting key functional and quality attribute requirements as scenarios. Each scenario exercises some architectural alternatives and each alternative represents a significant set of architectural decisions that must be made. The risks determined in this phase are focused on business strategy, the validity of the business goals, and business/IT misalignment. Strategic decisions on whether to develop a MVP are often made in this phase. Low-medium fidelity UI mockups or architecture spikes may then be created to further understand technologies, clarify business and innovation goals and requirements, and demonstrate value. A spike in agile methodologies is a miniature project, typically driven by a user story, aimed at answering a specific question. An architecture spike is similar, but is driven by a quality attribute question, most commonly regarding performance but it could, in theory, address any system quality [13]. In an architecture spike, a prototype is typically created to address the quality attribute question. It may be developed on the main branch, but architecture spikes are typically developed on a separate branch and (if successful) merged with the main branch.

Once past the VD phase, then the regular agile lifecycle starts. The business and innovation goal outputs of VD, as well as design constraints, concerns, and drivers from the Requirements Analysis (RA) phase are inputs for system design. In the RA phase, an analysis of the architectural requirements, expressed as quality attribute scenarios helps to identify additional risks: those of missing, conflicting, ambiguous, or under-specified architecture requirements. Quality attributes, such as performance, security, modifiability, and availability are the drivers of architectural decisions—not functionality. For this reason, the RA phase must focus on eliciting, validating, and prioritizing the quality attribute requirements, as this will guide subsequent architecture design and technology selection.

The Design phase of BDD performs architecture design, data modeling and technology selection in parallel [9][11]. The iterative process in the Design phase is based on reuse of well-known, assessed, and documented design primitives from the Design Concepts Catalog [11] (Step 6 of "Design" in Figure 1) which makes architecture design simpler, more capable, and more repeatable. Three components of the catalog are used in BDD's design steps: (1) reference architecture and frameworks; (2) design patterns, tactics and data models; and (3) a big data technology catalog. For example, when choosing a reference architecture for a data analytics application (Step 1), an architect could consult the catalog and see that the available options are: extended relational, pure non-relational, data refinery, and Lambda architecture. The architect could also consult the Design Concepts Catalog for specific technologies. For example, when choosing a messaging technology for component integration, the architect could choose among Apache Kafka, Amazon SQS, Apache ActiveMQ, or RabbitMQ. (An example Design Concepts Catalog may be seen at: http://smartdecisionsgame.com.)

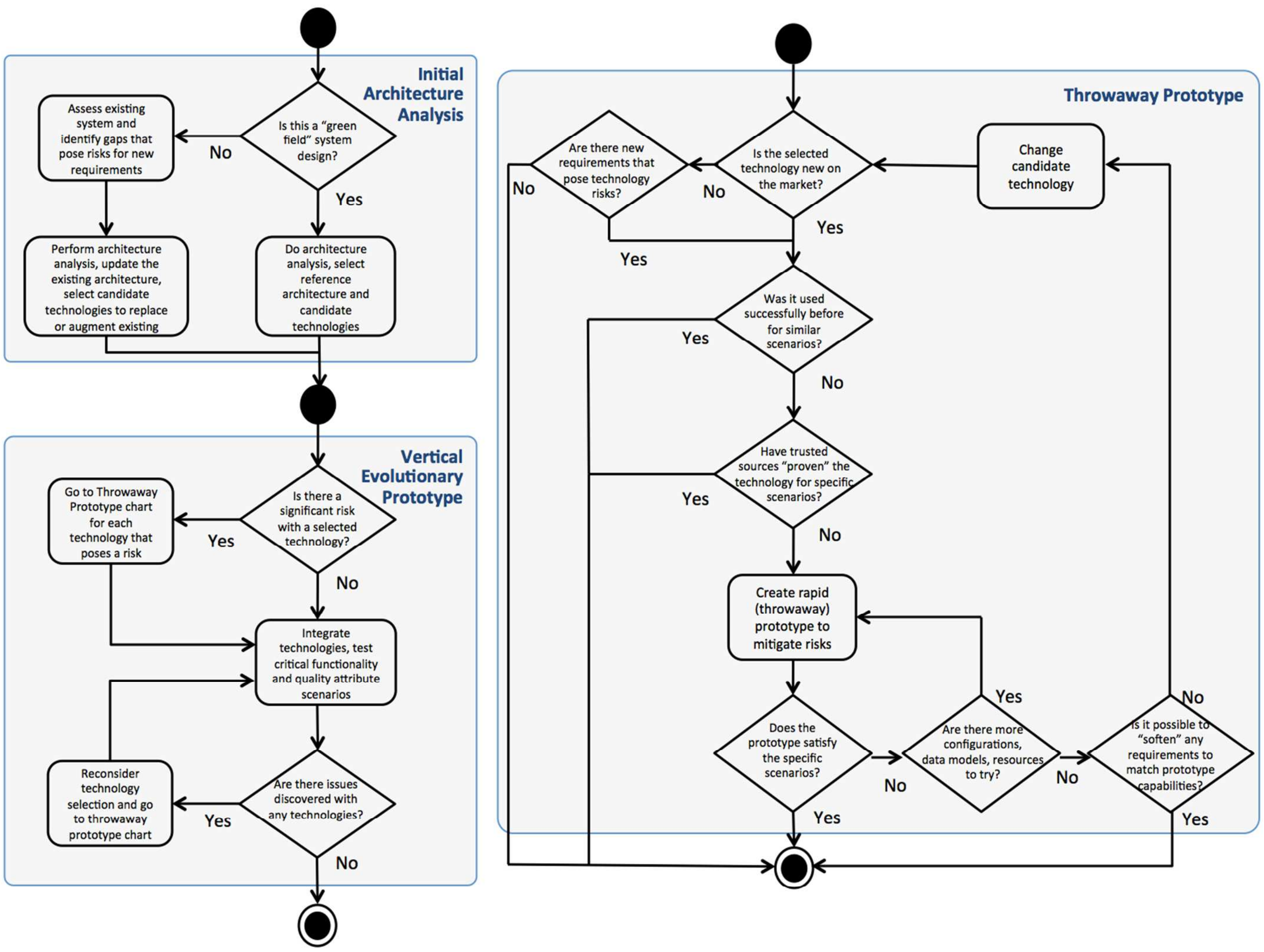
In Step 1 a reference architecture is chosen from the Design Concepts Catalog. Steps 2 and 3 of BDD—creating an architecture landscape and designing the big data data-flow diagrams (BD-DFD) are performed. (Details can be found in [7][9].) In steps 4-9, a system and a context diagram are decomposed and modeled in detail in a number of iterations. Each design iteration establishes iteration goals (Step 4) and then selects a quality attribute, driver, concern, or system component to focus on (Step 5). The selection of design concepts, such as patterns, tactics and frameworks, and selection of data models (based on inputs captured in the big data scenarios) are performed simultaneously in Step 6. In Step 7, the instantiation of architectural elements includes selection of databases, analytics platforms and other big data components. The technologies chosen for realizing conceptual components are critical architectural decisions and, in some cases, Steps 6 and 7 can only be achieved via experimentation, e.g., a spike.

Architecture analysis in the Design phase identifies risks concerning the architectural decisions made (or not made) relative to the high-priority scenarios. We do not attempt to fix the risks identified here; risk mitigation is a separate, follow-on activity. The discovered risks might be mitigated by changing the design, the requirements, or the business goals of the system. They might be mitigated by making additional project management decisions (e.g. retraining personnel, outsourcing portions of development, etc.). Or they might be mitigated by prototyping, experiments, more detailed analyses, or simulations. The risk mitigation technique that we focus on in this paper is prototyping.

In Step 8, BDD follows Attribute Driven Design (ADD) [1]. The design decisions made, and the rationale for these decisions, are documented. In Step 9 the design decisions made are evaluated based on iteration goals and measurements taken from any internal prototype or MVP. After a set of design iterations, a more complete analysis may be performed on the architecture to discover risks and identify tradeoffs before committing to a release (Step 10). In BDD, we employ the BITAM (Business and IT Alignment) method [8]—a light-weight architecture analysis method based on the ATAM [1]—to identify risks and, in particular, IT misalignment. At the end of each major development cycle or system release, an architecture analysis for identifying risks of future requirements, new business goals and strategies may be performed. In the Evaluation phase, processes similar to those in the VD phase will be performed for continuous exploration of big data value. Next we describe the RASP model, depicted on the right-hand side of Figure 1, and show how this complements the design activities of BDD.

Applying the RASP Model

Decisions regarding prototyping are critical in big data system development and hence should be made in a consistent, disciplined way. Technology *orchestration* is the core of big data system development, which means that such development relies heavily on architecture analysis and prototyping for risk management. The RASP model guides such analysis and prototyping activities. As mentioned above, the RASP model grew out of our collaborative practice research with SSV. Based on the lessons learned, we codified a set of questions that must be answered regarding the technologies being considered, and a decision-making procedure shown in Figure 2 that must be followed, before embarking on a (relatively costly) prototyping effort.



Initial Architecture Analysis
Is this a "green field" system design?
No
Assess existing system and identify gaps that pose risks for new requirements
Yes
Perform architecture analysis, update the existing architecture, select candidate technologies to replace or augment existing
Do architecture analysis, select reference architecture and candidate technologies
Vertical Evolutionary Prototype
Is there a significant risk with a selected technology?
Yes
Go to Throwaway Prototype chart for each technology that poses a risk
No
Integrate technologies, test critical functionality and quality attribute scenarios
Are there issues discovered with any technologies?
Yes
Reconsider technology selection and go to throwaway prototype chart
No
Throwaway Prototype
Is the selected technology new on the market?
No
Are there new requirements that pose technology risks?
No
Yes
Yes
Change candidate technology
Was it used successfully before for similar scenarios?
Yes
No
Have trusted sources "proven" the technology for specific scenarios?
Yes
No
Create rapid (throwaway) prototype to mitigate risks
Does the prototype satisfy the specific scenarios?
Yes
No
Are there more configurations, data models, resources to try?
Yes
No
Is it possible to "soften" any requirements to match prototype capabilities?
No
Yes

Figure 2: Strategic Prototyping Decision Flowchart

Employing standardized questions and decision procedures as well as architecture analysis methods helped us to plan prototyping systematically and strategically. For example, questions focused on whether the technology was new, whether we had successfully used it before, whether there were significant new requirements, whether there was objective evidence to substantiate claims of product performance, etc. The answers to these questions helped us to choose whether to prototype and how to evaluate the results of the prototyping effort. For example, an unsuccessful prototype might result in choosing a different technology, but another option is to attempt to "soften" some of the requirements. Our case studies exemplify each of these kinds of decisions.

The RASP model was followed in each of the nine cases. RASP was applied alongside the BDD method (and its associated Design Concepts Catalog) to systematically design and manage risk in each of these big data projects. As each project had different business and innovation goals, their risks were different. However, there emerged consistent risks themes in: technology selection, performance of new technologies, total cost of ownership, and integration of technologies.

Our case studies showed that *not* all projects needed to employ prototyping; architecture analysis alone sufficed in two cases where quick feedback on budget and schedule was required for business decision-making. Prototyping was strategically employed in the other seven cases and proved to be risk- and cost-effective. In each case, the effort for creating evolutionary and throw-away prototypes was much less time than horizontal prototyping, ranging from 2 to 6 weeks (although creating and evolving a MVP may take longer). Based on these case studies and our prior experiences, we have created and verified guidelines distilled from lessons learned for when to do architecture analysis, when to prototype, and under what conditions each specific class of prototype should be deployed:

1. Architecture analysis can be the only feasible option to make *early* decisions, and these decisions typically have project-wide scope. Then throwaway prototypes can be employed to make technology choices and demonstrate feasibility.
2. Architecture analysis alone is insufficient to prove many important system properties.
3. Architecture analysis complements vertical evolutionary prototyping: analysis helps to select candidate technologies, and prototyping validates those choices, and gets a working system in front of the customer quickly, which aids in eliciting feedback and prioritizing future development efforts.
4. An evolutionary prototype can be an effective risk mitigation strategy if it is implemented as a *skeleton*, i.e. the infrastructure into which components and technologies can be integrated [1]. This helps to mitigate integration risks, supports early demonstrations of end-user value, and aids in the understanding of end-to-end scenarios.
5. Vertical evolutionary prototyping can help answer questions about system-wide properties, but may need to be augmented with throwaway prototypes when requirements are volatile. When requirements change new architecture analysis and PoC prototypes may be needed (to help avoid falling into the "hammer looking for a nail" syndrome).
6. Throwaway prototypes, also known as *rapid prototypes* or *proofs of concepts* (PoC) work best when we need to quickly evaluate a technology, to see if it can satisfy critical architecture scenarios (usually related to quality attributes such as performance and scalability). A throwaway prototype is typically is used for narrow scenarios that focus on the evaluation of just one or two components.
7. The decision to create a MVP is a *business* decision more than one simply driven by technological risk. Typical business goals addressed include: getting early feedback from end users and updating the product accordingly; presenting a working version to a trade show or customer event; structuring the development process; checking team progress and alignment; or testing an outsourcing vendor using MVP as a pilot project.

Importantly, in each of our nine case studies, we did not move to create *any* prototype unless we had first determined that architecture analysis alone was inadequate to address risk.

Conclusion

Risk analysis has been advocated for controlling the evolutionary prototyping process [5] and our research has expanded on this position. Our research, departing from prior approaches to prototyping, demonstrates that an architecture-centric approach—the RASP model—can help makes risk management explicit, systematic and cost-effective. The need for *strategic* prototyping is critical for big data system development which involves high risk, high complexity and immature technologies. The RASP model builds on the BDD method—although it could, in principle, build on any architecture design method—offering guidelines for strategic prototyping, combining architecture analysis and prototyping methods. It provides a basis for value discovery with stakeholders, for reasoning about risks, for planning and estimating cost and schedule, and for supporting experimentation.

Prior studies [14] have attempted to understand how organizations decide whether or not to use prototyping and what environmental and organizational variables such as project size, developer, user, and organizational characteristics, affect the "proper" use of prototyping but did not have any significant findings. Others have attempted to create a risk list to guide risk management but the number of factors uncovered were large (over 100) rendering it impractical to use [2]. We believe our architecture-centric approach of using use cases and quality attributes scenarios and the prioritization of these scenarios provides a more intuitive, efficient and effective way of identifying, mitigating and continuously monitoring risks.

Specifically, our research contributes the Strategic Prototyping Decision Flowchart, and the associated guidelines distilled from lessons learned from nine extensive case studies to help make the RASP repeatable and prototyping cost-effective. For a big data system to be successful, architects need to plan and manage prototyping in a strategic, disciplined fashion. The RASP model helps achieve that end. However, a design or development method, no matter how thorough, cannot guarantee success. The architectural agility created and the architecture analyses performed in employing the RASP model rely on the architect's training and discipline.

References

[1] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice, 3rd edition*, Addison-Wesley, 2012.

[2] L. Bass, R. Nord, W. Wood, D. Zubrow. "Risk Themes Discovered through Architecture Evaluations," Technical report CMU/SEI-2006-TR-012, September 2006.

[3] S. Bellomo, I. Gorton, R. Kazman, "Insights from 15 Years of ATAM Data: Towards Agile Architecture", *IEEE Software*, 32(5), 38-45, 2015.

[4] B. Boehm. *Software Risk Management*. IEEE Computer Society Press, 1989.

[5] R.L. Baskerville, J. Stage. "Controlling Prototype Development Through Risk Analysis", *MIS Quarterly*, 20(4), 481-504, 1996.

[6] H. Cervantes, P. Velasco, R. Kazman, "A Principled Way of Using Frameworks in Architectural Design", *IEEE Software*, 46-53, March/April 2013.

[7] H-M Chen, R. Kazman "Architecting Ultra-Large-Scale Green Information Systems", *Proceedings of GREENS'12 at ICSE 34*, June 2012.

[8] H-M Chen, R. Kazman, A. Garg, "Managing Misalignments between Business and IT Architectures: A BITAM Approach," *Journal of Science of Computer Programming*, 57(1), 5-26, 2005.

[9] H-M Chen, R. Kazman, S. Haziyev, "Agile Big Data Analytics Development: An Architecture-Centric Approach", *Proceedings of HICSS 49*, (Kauai, Hawaii), 2016.

[10] H-M Chen, R. Schütz, R. Kazman, F. Matthes, "Amazon in the Air: Innovating with Big Data at Lufthansa", *Proceedings of HICSS 49*, (Kauai, Hawaii), 2016.

[11] H-M Chen, R. Kazman, S. Haziyev, O. Hrytsay, "Big Data System Development: An Embedded Case Study with a Global Outsourcing Firm," *Proceedings of BIGDSE'15 at ICSE 37*, 2015.

[12] R. Elmasri, S. Navathe, *Fundamentals of Database Systems*, 6th edition, Addison-Wesley, 2010.

[13] T.C.N. Graham, R. Kazman, C. Walmsley, "Agility and Experimentation: Practical Techniques for Resolving Architectural Tradeoffs", *Proceedings of ICSE 29*, 2007.

[14] B. Hardgrave, R. Wilson, K. Eastman, "Toward a Contingency Model for Selecting an Information System Prototyping Strategy" *Journal of Management Information Systems*, 15(2), 113-136, 1999.

[15] H. Simon, "The Architecture of Complexity," *Proceedings of the American Philosophical Society*, (106: 6), 1962.