Pragmatic Speculation
(…or, "what the @#%# is a Technical Use Case"?)

By Patrick Christiansen

A frequent challenge in IT organizations of scale is synthesizing the requirements of multiple projects taking place in parallel to reach an efficient rendering of overall "platform" needs. Beyond the core business features, the less obvious needs encompass common components, services, applications infrastructure (integration, security, instrumentation, etc.), and operational infrastructure (network, servers, storage, etc.). Approached in a project-by-project manner, the constituents responsible for each of the above concerns are rarely enabled to effectively organize the work and to deliver shared services. In business terms, IT ends up working against itself and reduces the ability to improve predictability while managing cost.

A method that I have proposed and utilized at many organizations involves a pragmatic meta-analysis step to increase the level of shared understanding and drive "shared speculation" of the platform needs mentioned above. A coined (and certainly non-unique) term involved in this exercise is the "technical use case". This is strictly a general terminology usage; this is not referring to a specific RUP or UML deliverable. In performing this within an organization, an existing requirement or design artifact (RUP or otherwise) is usually tailored to this exercise to represent the technical use case in the most consumable fashion for that organization.

The Setup

ABC Company has several projects executing in parallel that have some level of relationship. This relationship could be primary in terms of related business functionality, or it could be more obscure, such as having a desire to co-locate the hosting of the services on the same infrastructure. The key premise is that within the same 6-month to 12-month window, the company is looking to deliver on these projects while also *improving* the long-term predictability and cost factors of the overall environment.

The Challenge

Within each individual project, a development methodology is in place to understand the key business requirements (scenarios or use cases), and drive these to implementation. The predominance of project controls and deliverables continue to narrow the scope to insure delivery, and do not look outside the project for any substantial analysis. The common approaches to receive influential input from outside the projects (via reviews, governance, standard patterns, standard platforms, etc.) have diminishing returns if the environment around the project is immature or is changing rapidly; the instability equates to a lack of productive guidance or constraints for the project other than time and money. This common situation leads to many unproductive "catch-22" or "chicken-egg" discussions (how can I determine x without the requirements, and

how can I deliver requirements when I don't know what you are looking for…). The more catch-22s presented that create gridlock, the more self-sufficient each project is driven to be, and the more wheels are ultimately re-invented or delivered side by side.

One Approach: Pragmatic Speculation

Assuming that each project has established basic business scope and requirements (whether or not there are actually business use cases), a synthesis needs to occur to analyze the common characteristics across projects. The "technical use case" is one potential tactic to facilitate this analysis. The primary steps in creating a technical use case include:

- Select a finite sample of business scenarios or use cases that represent the essence of the combined project functionality for a consistent time frame.
- Consolidate the characteristics of these use cases, from both a functional and non-functional perspective. These characteristics should be discovered by questions provided by key subject-matter experts.
- The outcome of consolidating these characteristics is the technical use case (or set of technical use cases). This deliverable should include both diagram and descriptive statements. Key characteristics to capture include:
    - Users (key roles, geographic locations, user counts, key types of transactions)
    - Transactional Usage (for example read/update transaction types, assumed volume range)
    - Data (key object/entities, data stores, confidentiality characteristics)
    - Impacted/integrated application and systems mapped by assumed transactions
    - Stakeholders who have a stake in the outcome of the project such as business owners, customers, development and operational teams.
    - Relationships and integration points with other use cases.
    - Other key attributes as determined by the team
- An example of how the consolidation of the above characteristics can highlight critical path items earlier, and allow less project-specific discovery for specialty resources:
    - By understanding the nature of the users, locations, transaction types, and data, the need for security mechanisms to enable information exchange integrity, confidentiality, and availability can be discovered. For this specific area of requirements, to a certain extent, only the "highest common denominator" needs to be known of the users and the sensitivity of the data to provide basic security and integration design requirements. Typically, each project would be providing all requirement details to enable this discovery, while an efficient and effective design can be based on the consolidated

view. This does not replace the need to validate project-specific needs against the design assumptions, but aids in establishing the design assumptions themselves - and to see the forest through the trees.

- *Note: these consolidated characteristics are representing the combined use case scenarios, not independent scenarios. If business use cases are viewed to provide 75% functional requirements vs. 25% non-functional, this exercise is exactly opposite, providing much more focus on the non-functional (where "catch-22" situations are much more common). The resulting technical scenario is the highest common denominator of what is needed to support the business requirements. The purpose is to create a shared speculation of the necessary framework needed for the projects, not to perform or to replace the need for project-level analysis. This often must be done iteratively and in a time-boxed manner, to avoid the inevitable imperfection of requirements preventing educated speculation and reasonable risk-taking (see final note regarding approach caveats for more on this point).*

Following the creation of the technical use case, the representative subject matter experts perform architectural analysis (again, iteratively and time-boxed) and work together to produce a consistent recommendation for the design assumptions to enable and constrain each project. This provides the assumed target state that all projects and constituents understand, and links broad architectural vision to specific delivery assumptions, including priority and scope for common components, services, applications infrastructure (integration, security, instrumentation, etc.), and operational infrastructure (network, servers, storage, etc.). The approach also allows the pragmatic staging of a long-term vision over time, insuring project delivery needs are met but also effectively constrained to balance short and longer term needs.

Specifically, to make the recommendation real to the project teams, "mid-level" architectural views of a time-period or phase end-state are produced that show the combined project implementations (at each layer such as business, application, data, app infrastructure, and ops infrastructure). To clarify what is meant by "mid-level" detail, an example within the operational infrastructure view: to illustrate the processing platforms, the diagram would show the significant server, network, and storage constituents that must work together to deliver the combined technical use case(s), and specifically maps which services, components, and/or applications are hosted within a type of server "farm" (such as Unix, Wintel, etc.). This view would *not* attempt to estimate number of servers or specific configuration details, but provides a shared idea of platform growth, necessary integration connectivity to other platforms, challenges or contradictions posed by independent project mandates, and specific speculation about co-location of services (allowing a staffing/sourcing plan to be developed). Also, assumptions of supported versions of products should be exposed, as this is often a non-obvious point of surprise when purchasing products (both

applications run on this OS or DBMS, yet we discover that they require different versions…how will we plan on handling this? Different versions, forced upgrade, forced downgrade? Any of these answers may work, but exposing this issue early provides the engineers involved with the visibility necessary to drive to effective answers. Independent projects may seem to be able to provide this information, but usually cannot in a manner that supports broader decision-making.

If an organization is serious about the objectives of enabling delivery while *improving cost and predictability*, there is a necessity to utilize advanced resource sharing techniques from a technology perspective (SOA and shared services, virtualization, etc.) and from a staffing perspective (skill set pooling, sourcing strategies, etc.). Effective execution of these techniques will seldom be achieved without a tangible, operational level of shared vision *staged over time*. Done well, this exercise can provide each independent project the context required to make decisions that have a complementary or neutral affect on related projects (avoiding - or at least understanding and vetting - contradictory decisions). As projects execute, the target state views should be updated iteratively as significant assumptions are validated or invalidated (the speculated state proves true or false) and inevitable adjustments must be made. An additional frequent by-product is that mini-projects are started to offload work that was buried within the original multiple projects, increasing the ability to have a focused team working on high-risk items earlier, and not overloading high-cost specialty resources with meetings and business requirements that do not ultimately affect their scope of work.

Caveats

A few common issues exist with carrying this approach too far. Implicit in the resulting recommendation is a risk management statement to insure reasonable dependencies exist without jeopardizing business delivery (avoiding big-bang implementations or delaying key business benefits). Also, speculation implies that the requirements are partially known, not fully known. Therefore, a time-box approach is usually required to avoid analysis/paralysis and take reasonable and educated risks. It is more valuable to create the shared vision to validate than to wait for every technical detail.

Timing

There tends to be a natural rhythm dictating when it is a high-value time to undergo this meta-analysis activity; generally it is either a cyclical activity to drive budget and project planning, or it is a natural activity within a large program delivering multiple projects over time. Utilizing this analysis technique, work can be more efficiently organized, increased purchasing leverage can be exercised with software, hardware, and sourcing vendors, vendors and staff are better positioned to deliver, and the predictability and cost of delivery ultimately improves.