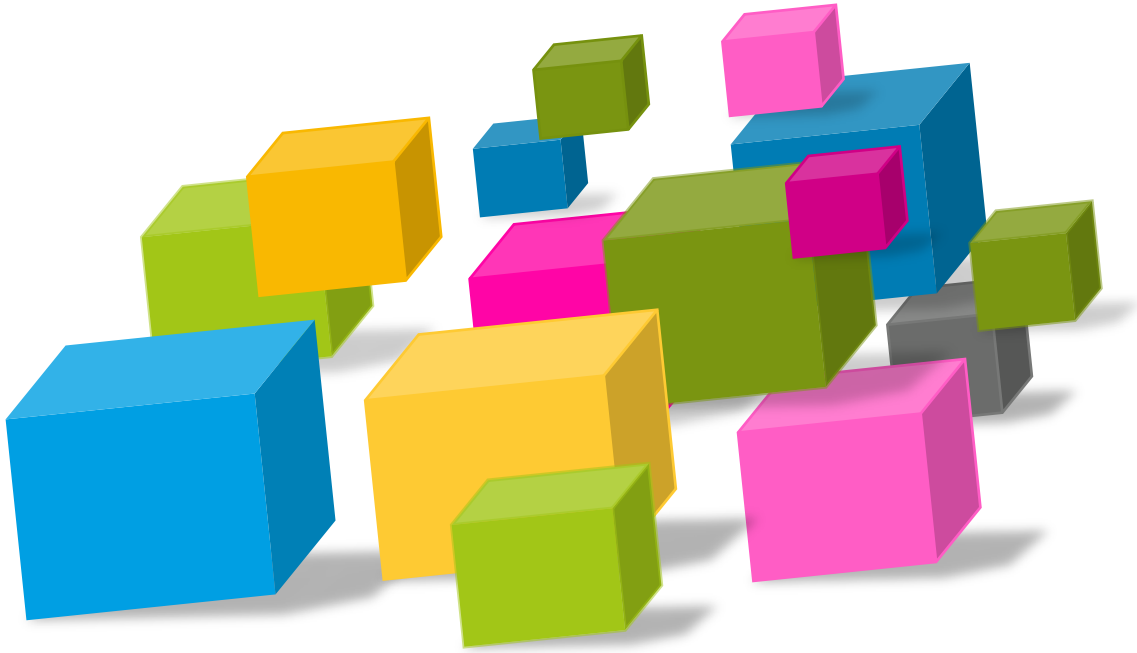
Risk and Prototyping

Kim Horn

2015



Risk

What is a Risk?

- A risk is a possible event or condition that can result in a missed target:
  - Requirements (e.g. incomplete, ambiguous, conflict between stakeholders, wrong type)
  - Schedule (e.g. cannot be estimated, it cannot be met, project will be over-run, no resources)
  - Quality (e.g. poor performance, crashes, errors)
  - Budget (e.g. way over budget)
- Risks can be technical or non-technical;
- **Risk management is a project management function**

Why Do We Care?

- Risks drive project planning
- What do we focus on first
  - Do we plan a prototype, plan investigate alternatives
- Risks drive technical focus
  - Doing the hard parts first!
- Risks can cause the project to fail!
- Too expensive to mitigate risks at the end;
- Uncovering and mitigating risk is a bridge between requirements and architecture

Uncovering Risks

- Experience, e.g.
  - Most of the team is new to Java
  - This is our first real project with Technology-X
- Control, e.g.
  - The date was defined by the client
  - Our subcontractor is on the critical path
  - Other projects have precedence over team members, e.g. fix major bug
- Resources, e.g.
  - Critical resources not identified
  - We have to be able to hire for construction
  - The business analysts should be 100% available in time for start of project
- Uncertainties, e.g.
  - What are the systemic qualities characteristics associated with this legacy system or external integration;
  - The User Interface may not be usable, or work in practice
  - The GUI function cannot be visualized or understood, it may not be able to navigate effectively to make it useful.
- Complexities, e.g.
  - This component is particularly complex; no skills in team;
  - Integration that involve complex and poorly defined APIs

Uncovering Risk (continued)

- Review the requirements, are there unidentified risks:
  - Particularly service level requirements: is each a slam dunk? Are they known ?
  - Conflicting requirements
- Interview key stakeholders
  - Are they on the same page and in agreement
- Public and project meetings
- Open and anonymous input
  - Does anyone want to submit an anonymous risk?

Risk and Requirements

Example:

- Requirement 1: Keep data private to preserve business reputation
- Requirement 2: Transfer financial data over the Internet
- Risk: Conflicting requirements
  - Several transactions involve the transference of sensitive data over the Internet
  - Security may impact performance

Questions to Ask

- If all these risks were resolved, would you bet on success?
- Have you talked to all key stakeholders?
- Is the risk articulated in a way that a clear mitigation plan can be defined?
- Has the real root cause been found?
- Are there any unknowns?

Risk Mitigation

- Confrontation
  - Prototyping (Next Section)
  - Testing
  - Project planning
- Alternative plans
  - Use different technology
  - Change or reduce requirements
  - Changing surrounding business process
  - Externalise (Outsource, Insurance)
- Acceptance
  - A high risk is accepted and business is wiling to gamble the whole project or its quality. Risk/Reward is high enough.

Risks and Iterations

- **Risk can be addressed early**, making early iterations more difficult.
  - In many projects the early iterations may address architecture and mitigate architectural risks.
  - Getting the foundation working is more important than features.
  - The hard issues are addressed first.

- **Risks can be pushed out** making early iterations easier. However earlier work may be throw away when risks are finally addressed.
  - Significant business value may be obtained by delivering something early, accepting risk to a later date.
  - The hard issues are addressed later.
  - However, this does not mitigate risk but may mean the whole endeavour is at risk, if later wok can resolve risks.

Prototyping

Prototyping

- Motivation for prototyping
- Pitfalls of prototyping
- Prototype Types
- What is a Proof of Concept ?
- Prototypes have architecture and design.

Motivation for Prototyping

- User Interface
  - Clarify requirements (look and feel)
  - Mitigate requirements risk
  - Demonstrate early progress
  - Determine usability / accessibility / quality issues
- Architectural
  - Mitigate technical risk
  - Create a stable baseline
  - Validate key architectural concepts
  - Test if quality can be provided: performance, latency etc
  - Provide examples for developers
  - Increase productivity of teams
  - Understand technology

Pitfalls of Prototyping

- Lack of focus (what is prototype for?)
  - No clear goals, no hypotheses.
  - No testable outcome
  - Unmanaged expectations
- Mistake user interface design for requirements definition.
- Throwaway becomes evolutionary (It is really a first iteration).
- Prototyping does not address the real business problem, but usually, too early, provides a small part of the technical solution.
- **Emergent Requirements:** Design and build will change requirements, discussed above. So a prototype unless very well defined and narrow, may need to be re-built, retested in delivery.
- **Tunnel Vision:** the prototype excludes other important features in the final system as work has been focused on tuning a small set of requirements.
- **Prototype Paralysis:** they can take a long time, include too much detail, debates, endless testing.
- **Insufficient Analysis:** prototyping is planned as quick and analysis is not done properly. As stated the analysis happens with building a real working system, not upfront.
- **The Artificial Reality problem:** the plausible presentation of an implausible design. A good artist or modeler can make most any prototype design look like it will work.

Pitfalls of Prototyping

- Early prototypes are too early, Houde and Hill say: *"It is common practice to build prototypes in order to represent different states of an evolving design, and to explore options. However, since interactive systems are complex, it may be difficult or impossible to create prototypes of whole design in the formative stages of a project."*
- Throw away prototypes are too expensive to be thrown away, so we build the poor quality code into the real system.
- Prototype cuts corners to deliver fast but this poor quality 'evolves' into working systems without fixing the cut corners.
- While the prototype is being built the real system is on hold.
- Prototypes are not the real system. Reality often means a prototype missed the point. One part of the system is prototyped other important parts are ignored.
- Prototype hides a real Iteration 0; to hide the real cost of development or portray a false sense of progress.
- Prototype approach used when not appropriate, root cause analysis not done. E.g. we have no experience or do not understand requirements, or have no idea what 'our' user/customers want.
- Prototypes are not self explanatory and looks can be deceiving. Clarifying what aspects of a prototype correspond to the eventual artefact, and what don't, can lead to confusion.

Prototypes are not Rapid

Prototyping, as a method, was the basis for Steve McConnel's book on RAD (Rapid Application Development) process *"Rapid Development: Taming Wild Software Schedules,."*

"Since rapid application development is an iterative and incremental process, it can lead to **a succession of prototypes that never culminate in a satisfactory production application**.....

Furthermore, there is a phenomenon behind a failed succession of unsatisfactory prototypes: End-users intuitively and primarily look to the Graphical User Interface (GUI) as a first measure of software qualities. The faster they "see" something working, the more "rapid" they perceive the development cycle to be. This is because the GUI has a strong natural directly viewable presence, while other problem domain code must be deduced or inferred, going largely unnoticed."

Types of Prototypes

	Throwaway	Evolutionary
User Interface	Demonstrate and/or validate concept or "look and feel" with users.	Same goal as throwaway, but intended to evolve into system.
Architectural	Explore and/or validate key internal architectural concepts.	Validate architecture through incremental development.

- Evolutionary Prototypes are really a first iteration

Prototype Realization (See Prototype Template)

Prototyping is itself risky: Focus prototypes only on specific tasks, and clearly define:

- Requirements
- Risk Areas and Coverage
  - Risks Addressed
  - Architecturally Significant Use Cases
  - Goals and Success Metrics
- Prototype Technical Approach
  - Constraints
  - Coverage
  - Architecture
  - Technical Options Addressed
  - Test Data Sets
- A clear Testable Hypothesis / Goal
- Well defined and Measurable Metrics for Success

The Proof of Concept (POC) Vs Prototype

"Proof of concept (POC) is evidence obtained from a pilot project, which is executed to demonstrate that a product idea, business plan, or project plan is feasible." projectmanager.com

- However, this just means the project has risks; and that these will be confronted:
  - As a result a prototype is built to test the risk can be mitigated.
  - This may be a separate project or an early iteration of a project.
- Above also suggest:

  " a POC is different to a prototype as the later is done to help visualize how a product will function in the real world. It shows design, navigation, layout, etc. Therefore, a proof of concept shows that a product idea can be made, and the prototype shows how it's made."
- If visualisation of function is not a risk then a POC/Prototype is not required.
- The Risk based approach does not make this artificial distinction distinction between a POC and a Prototype; they both are used to mitigate risks.
  We will only use the term Prototype.

The Prototypes are Architected and Designed.

- Prototypes can address Architectural Risks, therefore an Architecture is required.
- Other prototypes are working systems so need to be architected and designed.
  - Certain Qualities are Required for the prototype
  - Other qualities are not. These qualities are traded off in prototype in a way that is not with the real system tradeoff; these need to be well understood. Can they impact the final system too ?
- You Need a System Architecture for the Prototype

Conclusions

- Risk management is a project management function;
- Risks can ruin a project; Too expensive to address later;
- Risks can be accepted, confronted, or tackled by alternates;
- Prototypes address risks head on;
- Prototypes should be thrown away, not used; Evolutionary approaches are too hard to control; inject new risks;
- Prototypes should test a well defined hypothesis;
- Prototypes do not solve poor process, e.g. undefined goals or requirements;
- Prototypes are not an SDLC or process – they are part of the process/project, often an Iteration 0;
- Prototypes require resources and do have a time and cost;