Project Risk Reduction

Dr. James A. Bednar
jbednar@inf.ed.ac.uk
http://homepages.inf.ed.ac.uk/jbednar

Dr. David Robertson
dr@inf.ed.ac.uk
http://www.inf.ed.ac.uk/ssp/members/dave.htm

Project Risk Reduction

If a system is safety or business critical we expect to identify common risks and their remedies before deploying the system. The same should be true for SW design processes.

One way of doing this is through risk reduction patterns, which identify common sources of project risk and suggest how to reduce them. The goal is to be able to anticipate and handle risks gracefully, minimizing the danger to your project.

Typical Sources of Risk

- Imperfect knowledge of the problem
- Teamwork difficulties
- Lack of productivity
- Ambiguity over ownership
- Distractions
- Training new team members
- Depending on new, untested technologies
- Depending on external components out of your control
- Falling behind competitors working on similar projects

Risk Reduction Patterns

These all are taken from `members.aol.com/acockburn/riskcata/riskbook.htm`

- Knowledge: Clearing the Fog
- Knowledge: Early and Regular Delivery
- Knowledge: Prototype
- Teaming: Holistic Diversity
- Productivity: Gold Rush
- Ownership: Owner per Deliverable
- Ownership: Function versus Component
- Distractions: Someone Always Makes Progress
- Distractions: Team per Task
- Training: Day Care

Knowledge: Clearing the Fog

If you don't know the issues well enough to put together a sound plan, then try to deliver something (almost anything). Just trying to do that tells you what the real issues are. Do this when:

- You need more knowledge to proceed, but
- You have to move forward into the project

Knowledge: Clearing the Fog Example

You are considering a serious project using a totally new language or technology. You don't know whether to proceed or how to size the project. So, run a carefully instrumented, mini-version of the project. Collect data on staff learning rates, productivity, technology effects. From this data, you can extrapolate the total effect to your proposed project.

Cures versus Overdose

Cures:

- May discover what issues you need to address
- Basis for creating first project plan

Overdose:

- May waste effort in clearing the fog without making real progress
- Fog clearing becomes an aim in itself

Knowledge: Early and Regular Delivery

You don't know what problems you will encounter during development, so deliver something early to discover what you don't know you don't know. Deliver regularly and improve each time. Do this when:

- You are unsure about part of your development process
- You want to improve and optimize your process

Knowledge: Early and Regular Delivery Example

Your boss tells you that the executives only need to see a release every 10 months. You decide to create internal releases at 4 months and 7 months. This ensures that you have identified and reduced the risks for the 10-month delivery.

Cures versus Overdose

Cures:

- If you hit an unexpected problem you have lost no more than the internal delivery period
- You may be able to iron out minor problems in the next cycle
- A way of gathering solid data early
- Boosts morale if releases make progress

Overdose:

- Cycle too fast and setup/test may eat up your time
- Need effort to monitor fast cycles
- Saps morale if releases are chaotic

Knowledge: Prototype

You don't know how some design decision will work out, so build an isolated solution to discover how it really works. Do this when:

- You are designing a user interface, or
- You are trying a new database or network technology or
- You are dependent on a new, critical algorithm

Knowledge: Prototype Example

You are designing a user interface but probably will not get the design correct on the first try. You create a paper prototype in a few hours, or a screen prototype in a few hours or a day, or a rigged demo using fixed data. You show this to the users to discover missing information.

Cures versus Overdose

Cures:

- Small effort for (perhaps) big gains
- Produces hard evidence

Overdose:

- Can develop a “perpetual prototyping” culture — design keeps shifting because it’s easy to see what you don’t like, but hard to know when you are done
- Other teams can end up on hold waiting on prototypes to be approved

Teaming: Holistic Diversity

Development of a subsystem (a set of functions) needs many skills, but people specialize, so create a single team from multiple specialities (e.g. requirements gathering, UI design). The team should all be able to meet in person, and should be evaluated as one group. Do this when:

- People seem to be doing “throw it over the wall” (a.k.a. “passing the buck”) development
- Teams are grouped only by speciality
- People are communicating mainly by writing
- Teams do not appear to respect each other

Teaming: Holistic Diversity Example

Those who can do the requirements gathering and analysis interview people, and investigate interfaces and options. They communicate the results rapidly, face-to-face, with the people who navigate the class library and design classes and frameworks. Teams are formed consisting of a combined requirements analyst with a few program designers. These move rapidly through the design. They have no internal deliverables, but create the deliverables as required for communication between teams. Most of the communication is verbal.

Cures versus Overdose

Cures:

- Fast, rich feedback on decisions within teams
- Reduces risk of people protecting their specialities against other specialities
- Helps deal with shortage of multi-skilled individuals
- Everyone “designs” in some way

Overdose:

- 1-person teams can’t master all the specialities
- Too-large teams get bogged down in time-lagged chat
- Needs subtle coordination
- People in different teams may blame each other

Productivity: Gold Rush

You don't have time to wait for requirements to settle so start people designing and programming immediately, and adjust their requirements weekly. Do this when:

- You want design with care, and
- To avoid redoing work, but
- You need the system fast

Productivity: Gold Rush Example

You start with a rough set of requirements. The designers quickly get ahead of the requirements people, who are busy in meetings trying to nail down details of the requirements. If the designers wait until the requirements are solid they won't have enough time to do their work, but they can guess what the requirements would be like without knowing final details, so they start design and programming right away. The requirements people give them course corrections after each weekly meeting. The amount of time it takes to incorporate those mid-course alterations is small compared to the total design time.

Cures versus Overdose

Cures:

- Downstream people can start on the obvious parts of their work
- Frequent meetings help guessing about the rest

Overdose:

- Downstream people may get ahead of the stability of the upstream decisions, so have to redo more work
- In the extreme, final rework gets greater than the total development time

Ownership: Owner per Deliverable

Sometimes many people are working on it, sometimes nobody so make sure every deliverable has exactly one owner. Do this when:

- You detect a "common area" (chaotic updates with multiple ownership), or
- You detect an "orphan area" (no ownership), or
- Multiple teams are working on one task, or
- One person is working on many tasks

Ownership: Owner per Deliverable Example

You repeatedly find that there is no one person who answers for the quality and currency of the design blueprints, the quality an consistency of the user interface, the program code, or the performance of the system.

Cures versus Overdose

Cures:

- Resolves issues of who should do things like maintenance
- Helps establish internal integrity of components

Overdose:

- Ownership of everything on the project can be seen by some people as wonderful, and others as a nuisance
- Ownership may create conflict, so conflict management saps the team's energy
- Productivity can get stalled when owners go missing

Ownership: Function versus Component

If you organize teams by components, functions suffer, and vice versa, so make sure every function has an owner and every component has an owner. Do this when:

- Your teams are organized by function or use case, with no component ownership, or
- Your teams are organized by class or component with no function, use case, or user story ownership

Ownership: Function versus Component Example

The project starts out with teams centred around classes or components. At delivery time, the end function does not work. Each team says, "I thought you were taking care of that. It doesn't belong in my class."

Cures versus Overdose

Cures:

- Consistency and quality of functions, not just components
- Assists sharing of components across teams

Overdose:

- Possible friction between function and component owners
- Interconnections between functions and components can get confusing
- Ownership by function turns components into commons
- Ownership by components turns functions into orphans

Distractions: Someone Makes Progress

r Distractions constantly interrupt your team's progress so, whatever happens, ensure someone keeps moving toward your primary goal. If you do not complete your primary task, nothing else will matter. Therefore, complete that at all costs. Do this when:

- Non-primary tasks are dominating the team's time
- Many people complain of distraction

Distractions: Someone Makes Progress Example

In the ancient Greek story, Atalanta was assured by the gods that she would remain the fastest runner as long as she remained a virgin so she agreed to marry only the man who could beat her in a foot race. The losers were to be killed for wasting her time. The successful young man was aided by a god, who gave him 3 golden apples. Each time Atalanta pulled ahead, he tossed an apple in front of her. While she paused to pick up the golden apple, he raced ahead.

Cures versus Overdose

Cures:

- If at least someone is making progress on the primary task then you are somewhat closer to your final goal
- Allows some attention to every task, including small diverting ones

Overdose:

- You may eventually get into trouble for not adequately addressing the distractions
- Too many distractions suggest that there is a deeper problem

Distractions: Team per Task

A big diversion hits your team so let a sub-team handle the diversion, the main team keeps going. Do this when:

- Requirements gathering is taking longer than the schedule can allow, or
- The version in test needs attention, but so does the version in development, or
- Your people say “We have too many tasks, causing us to lose precious design cycles”

Distractions: Team per Task Example

You have holistic design teams but each person in the team is doing requirements, analysis, design and programming. But requirements meetings become frequent and it is difficult to switch mode from these meetings to programming, so little programming is being done. You allocate members of each team to specialize in requirements for a while, freeing others to concentrate on programming.

Cures versus Overdose

Cures:

- Allows prioritization of tasks within teams
- Helps focus on primary goals

Overdose:

- You may eventually have one-person teams
- Enforced splits may break synergy of teams

Distractions: Sacrifice One Person

A smaller diversion hits your team so assign just one person to it until it gets handled. Do this when:

- Diversions as “Team per Task” but
- These are small enough to be handled by individuals and
- They couldn’t be handled easily by allowing switching between tasks

Distractions: Sacrifice One Person Example

Odysseus has to get his ship past Scylla and Charybdis. Scylla is a six-headed monster, guaranteed to eat six crew members, but the rest would survive. Charydbis is a whirlpool guaranteed to destroy the entire ship. Odysseus sacrifices six people to Scylla.

Cures versus Overdose

Cures:

- The main group of the team moves forward without the distraction
- Avoids loss of everyone's time in task switching

Overdose:

- The person assigned to the distracting task may be alienated
- If you keep sacrificing individuals you have no one on the primary task

Training: Day Care

Your experts are spending all their time mentoring novices, so put one expert in charge of all the novices, and let the others develop the system. Do this when:

- You experts say "We are wasting our experts", or
- "A few experts could do the whole project faster", but
- You have to add several new people to an existing project

Training: Day Care Example

You have put 4 novices under each expert. The experts now spend the most of their energies training, halfheartedly. They are caught between trying to get the maximum out of their trainees and trying to do the maximum development themselves and neither develop the system, nor train the novices adequately. You institute an "apprenticeship" program in which novices are given a dedicated mentor for 2 weeks out of every 3 for 6 months.

Cures versus Overdose

Cures:

- Separates “progress” teams from training teams
- Allows selectivity of trainers
- Localizes impact of new recruits

Overdose:

- Can be viewed as “sacrificial”
- Your progress team can dwindle to zero

Summary

- All projects have risks
- Big risks need to be anticipated and addressed
- Risk patterns help you notice and deal with common project risks
- Overdoing the correction adds another risk
- Required reading:
  `http://members.aol.com/acockburn/riskcata/riskbook.htm`