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 software 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.

Sources of risk include:

- Imperfect knowledge of the problem.
- Teamwork difficulties.
- Lack of productivity.
- Ambiguity over ownership.
- Distractions.
- Training overheads.

0.1 Risk Reduction Patterns

One way of establishing a body of shared experience in risk reduction is by documenting standard patterns of project management which have been known to reduce risk in the past. The ones in this lecture all are taken from `members.aol.com/acockbu` Although each pattern has the potential to reduce project risk there is also the possibility that we can "overdose" on it, leading to unwelcome side-effects.

0.1.1 Knowledge: Clearing the Fog

You don't know the issues well enough to put together a sound plan, so try to deliver something (almost anything); this tell you what the real issues are.

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

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:

- 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.

0.1.2 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

- You are unsure about some part of your development process.
- You want to improve and optimise.

Example: You have been told by your boss 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:

- 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.

0.1.3 Knowledge: Prototype

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

- 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

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:

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

Overdose:

- Can develop a "perpetual prototyping" culture.
- Other efforts may rely unduly on prototype success.

0.1.4 Teaming: Holistic Diversity

Development of a subsystem needs many skills but people specialise so create a single team from multiple specialities.

- "Throw it over the wall" development.
- Teams are structured by speciality or phase.
- Communication by writing.
- Lack of respect of one team for another.

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:

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

Overdose:

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

0.1.5 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.

- You want design with care and
- to avoid redoing work but
- you need the system fast

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:

- Downstream people can start on the obvious parts of their work.
- Frequent meetings help make best guesses about the rest.

Overdose:

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

0.1.6 Ownership: Owner per Deliverable

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

- 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.

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:

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

Overdose:

- Ownership of every thing on the project can be perceived by some people as wonderful, and others as a nuisance.
- Ownership may create conflict, so conflict management saps the team's energy.

0.1.7 Ownership: Function versus Component

If you organise teams by components, functions suffer, and vice versa, so make sure every function has an owner and every component has an owner.

- Your teams are organised by function or use case, with no component ownership or
- your teams are organised by class or component with no function or use case ownership.

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:

- Ownership and consistency of functions, not just components.
- Assists sharing of components across teams.

Overdose:

- Possible friction between function and component owners.
- Can become lost in the interconnections between functions and components.
- Ownership by function turns components into shared areas.
- Ownership by components turns functions into orphan areas.

0.1.8 Distractions: Someone Makes Progress

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.

- Non-primary tasks are dominating the team's time.
- Common complaints of distraction.

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:

- 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 are a symptom of a deeper problem.

0.1.9 Distractions: Team per Task

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

- 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"

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 specialise in requirements for a while, freeing others to concentrate on programming.

Cures:

- Allows prioritisation of tasks within teams.
- Helps focus on primary goals.

Overdose:

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

0.1.10 Distractions: Sacrifice One Person

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

- 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.

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:

- 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.

0.1.11 Training: Day Care

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

- You experts say "We are wasting our experts" or
- "A few experts could do the whole project faster" but
- you have to add a batch of new people to an existing project.

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:

- Separates "progress" teams from training teams.
- Allows selectivity of trainers.
- Localises impact of new recruits.

Overdose:

- Can be viewed as "sacrificial".
- Your progress team can dwindle to zero.
- Can still proliferate training commitments.

0.2 Exercise

Look back at each of the projects summarised in the lecture on project failures. How do the problems we summarised for these failed projects mach to the risk patterns above.