Risk Management

- There are many threats to the success of your project.

- It is crucial to identify the worst risks at the outset of your project, and to take constructive action to mitigate the effects of those risks.

- Instead of planning assuming the best case, you should incorporate the ways things often go wrong into your plan.

- The goal is to be able to anticipate and handle risks gracefully, minimising 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

Named for the solution; not the risk.

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

[Examples from Alistair Cockburn’s Risk Catalogue]

Crucial skill is knowing when and how to apply them.
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

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.
Clearing the Fog

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
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 optimise your process

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.
Early and Regular 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
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

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.
Prototype

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
Holistic Diversity

Development of a subsystem (a set of functions) needs many skills, but people specialise, so create a single team from multiple specialities. 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” development
- Teams are grouped only by speciality
- People are communicating mainly by writing
- Teams do not appear to respect each other

**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.
Holistic Diversity

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
**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 to 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.
Gold Rush

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
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

Example: “In an orphan area, the class diagram often is out of date because it is no one’s job to update it. Find out who owns that segment of the class structure, and assign someone to own the diagram [...] In order not to waste too much of that person’s time, decide on the fewest occasions at which the class diagram must be current. Note that every piece of final documentation is a deliverable that requires ownership.” [A. Cockburn, Surviving Object-Oriented Projects: A manager’s guide, Addison-Wesley, 1998]
Owner per Deliverable

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, and conflict management saps the team’s energy
- Productivity can get stalled when owners go missing
Functionality versus Component

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

- 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, use case, or user story ownership

**Example:** “Project Reginald started out with teams centred around classes or components. At delivery time, the end function did not work. Each team said, “I thought you were taking care of that. It doesn’t belong in my class.” Later on in this project, each function had one responsible owner. That person had to negotiate between the teams to see if one team was willing to close the gap, or to create some extra, special code to close the gaps between the classes.”

[A. Cockburn, Surviving Object-Oriented Projects: A manager’s guide, Addison-Wesley, 1998]
Functionality versus Component

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
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. Do this when:

- Non-primary tasks are dominating the team’s time
- Many people complain of distraction
Someone Makes Progress

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
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”

**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.
Team per Task

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
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
Sacrifice One Person

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’ll have no one on the primary task
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

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.
Day Care

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
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 Readings

- Alistair Cockburn, *Project risk reduction patterns*


- E.H. Conrow, P.S. Shishido. *Implementing Risk Management on Software Intensive Projects*. IEEE Software May/June 1997. [http://dx.doi.org/10.1109/52.589242](http://dx.doi.org/10.1109/52.589242)
Suggested Readings

