Risk Management

Software Architecture, Process, and Management

The previous lecture looked at ways large projects tend to fail. This lecture looks at how we can attempt to avoid failure by recognising the risks that our current projects are subject to.

Risk

- Just as failure was a topic I enjoy, so too is *risk*
- As you may have been able to guess, risk is not something our evolved brains are particularly well suited to
- Why?
- Our brains evolved in a very different world to that which we now reside in
  - People lived in small groups
  - Rarely met anyone who was much different from themselves
  - Had rather short lives
  - Highest priority was to eat and mate **today**

Why

- We are less worried about dangers and risks that we believe are avoidable
- Even when statistics tell you otherwise
- This is why people drive cars, they believe themselves to be in control and that is enough to prevent their own untimely demise
- When you are in a plane, you have to accept that you have entirely given up any control, or illusion of control, you may have had
- Perhaps this is part of the reason we continue to commission very large-scale software

Risk Management

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

Risk Management

- It is crucial to identify the worst risks ~~at the outset~~ **throughout the lift time** of your project, and to take constructive action to mitigate the effects of those risks
- This is why, although we know it to be difficult, we still try to **estimate** and **measure**
- In the previous lecture we saw that many software project failures were caused by problems of management and external forces, hence it is not surprising that risk management strategies are directed towards project managers

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
- Examples from Cockburn's Risk Catalog, we will look at the following categories:
  - Ownership
  - Distractions
  - Training
  - Knowledge
  - Teaming
  - Productivity
- The pattern does not give detailed instructions, but is a general solution to the problems indicated
- Crucial skill is knowing when and how to apply them

Risk Reduction Patterns

- It is important to realise that the solutions are not a general best practice
- The solution is deployed only when it is recognised that the risk is evident

Ownership: Owner Per Deliverable

Risk Example Cures vs Overdose

- Sometimes many people are working on it, sometimes nobody, so make sure every deliverable has exactly **one** owner
- Ask and make sure that you can answer who is ultimately responsible for deliverable aspects, such as documentation
  - In particular, onerous tasks which tend not to get done if they are not **one** person's responsibility
  - It need not be the owner that actually does the work, just that see that it gets done
- This applies not just to code,but to documentation or any other deliverable

Ownership: Owner Per Deliverable

Risk Example Cures vs Overdose

- 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

Risk Example Cures vs Overdose

- You are working on a large-scale web-based project
- You repeatedly find that no one person answers for:
  - the quality and consistency of the user interface,
  - or the performance of the system

Ownership: Owner Per Deliverable

Risk Example Cures vs 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, and conflict management saps the team's energy
  - Productivity can get stalled when owners go missing

Ownership: Function versus Component

Risk Example Cures vs Overdose

- If you organise teams by components:
  - No one is responsible for any given function
  - Hence all teams may claim that the given functionality does not belong in their components
  - E.g. Validation of input, is often unclear
- If you organise teams by functions/use cases etc.:
  - Code structure tends to suffer
  - No one is responsible for the *cohesion* of any particular class, as a result code, is added as required by each function
- So, make sure every function has an owner and every component has an owner

Ownership: Function versus Component

Risk Example Cures vs Overdose

- 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

Ownership: Function versus Component

Risk Example Cures vs Overdose

- 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.”

Ownership: Function versus Component

Risk Example Cures vs 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

Risk Example Cures vs Overdose

- A general pattern, almost like an abstract sub-class
- With specialisations:
  - Team per task
  - Sacrifice One Person
  - Day-care

Distractions: Someone Makes Progress

Risk Example Cures vs Overdose

- 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

Distractions: Someone Makes Progress

Risk Example Cures vs 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

Risk Example Cures vs Overdose

- A specialisation of “Someone Makes Progress”
- 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

Risk Example Cures vs Overdose

- The external environment changes significantly, perhaps an unforeseen change in the law, or an acquisition of the customer company
- This changes requirements significantly, meaning many more requirements meetings
- 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

Distractions: Team Per Task

Risk Example Cures vs Overdose

- Cures:
  - Allows prioritisation of tasks within teams
  - Helps focus on primary goals
  - It is time-consuming to switch tasks
- Overdose:
  - You may eventually have one-person teams
  - Enforced splits may break synergy of teams

Distractions: Sacrifice One Person

Risk Example Cures vs Overdose

- Also a specialisation of “Someone Makes Progress”
- 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

Risk Example Cures vs Overdose

- You just finished a release candidate and are now receiving many small bug reports and feature requests
- It seems reasonable to allow each team member to work on the issues related to their concerns
- But many bugs overlap, are poorly categorised
- And it takes time to switch between debugging and implementing new code

Distractions: Sacrifice One Person

Risk Example Cures vs 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'll have no one on the primary task

Training: Day Care

Risk Example Cures vs Overdose

- Again a specialisation of "Someone Makes Progress"
- 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:
  - Your 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
  - It may be that both the delivered system and trained people are required results of your project

Training: Day Care

Risk Example Cures vs Overdose

- You have put 4 novices under each expert
- The experts now spend the most of their energies training, half-heartedly
- 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

Training: Day Care

Risk Example Cures vs Overdose

- 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

Knowledge: Clearing the Fog

Risk Example Cures vs Overdose

- If you do not know the issues well enough to put together a sound plan
- So, 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

Risk Example Cures vs Overdose

- You are considering a serious project using a totally new language or technology
- You do not 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

Knowledge: Clearing the Fog

Risk Example Cures vs 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

Risk Example Cures vs Overdose

- A specialisation of “Clearing the Fog”
- You do not 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

Knowledge: Early and Regular Delivery

Risk Example Cures vs Overdose

- Your boss tells you that the executives only need to see a release every 10 months
- You decide to create internal releases every 2 months
- This ensures that you have identified and reduced the risks for the 10-month delivery

Knowledge: Early and Regular Delivery

Risk Example Cures vs 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

Risk Example Cures vs Overdose

- Also a specialisation of "Clearing the Fog"
- 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

Risk Example Cures vs Overdose

- 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

Knowledge: Prototype

Risk Example Cures vs 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
  - *"Protoduction"*

Teaming: Holistic Diversity

Risk Example Cures vs Overdose

- Development of a subsystem (a set of functions) needs many skills, but people specialise
- Groups/teams tend to help those within their team and blame those not within their team
- 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

Teaming: Holistic Diversity

Risk Example Cures vs Overdose

- Do this when:
  - People seem to be doing “throw it over the wall” or “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

Risk Example Cures vs Overdose

- You are developing an application with a Model-View-Controller architecture
- So you have a team for each of the Model, View and Controller
- Some problems are inarguably a problem of one particular team
- But many problems will be shared, it is even possible to be in cycle of dependencies
- So, create a team consisting of at least one member from each of the original teams to make progress on such shared problems

Teaming: Holistic Diversity

Risk Example Cures vs 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

Risk Example Cures vs Overdose

- 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

Productivity: Gold Rush

Risk Example Cures vs Overdose

- 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 will not 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

Productivity: Gold Rush

Risk Example Cures vs 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

Pattern Vagueness

- Many of these patterns are necessarily vague
- The important skill is that of recognising when one of these applies
- Or indeed you face an undocumented risk

Don’t Forget

- We are not very good at assessing risk
- We’re also unjustifiably confident in our ability to assess risk
- We are also extremely impatient and prefer gratification now to later
- This may be a reason code-bases tend to fall into disorganisation
- There isn’t much you can do about this; these risk patterns may help
- Be prepared to be an unsung hero